A system for generation of near real-time feeds of user-customized hydrometeorology data-products from NEXRAD radar-data

Satpreet Harcharan Singh
University of Iowa

Follow this and additional works at: https://ir.uiowa.edu/etd
Part of the Electrical and Computer Engineering Commons

A SYSTEM FOR GENERATION OF NEAR REAL-TIME FEEDS OF USER-CUSTOMIZED HYDROMETEOROLOGY DATA-PRODUCTS FROM NEXRAD RADAR-DATA

by

Satpreet Harcharan Singh

A thesis submitted in partial fulfillment of the
requirements for the Master of Science degree
in Electrical and Computer Engineering
in the Graduate College of
The University of Iowa

December 2010

Thesis Supervisor: Associate Professor Anton Kruger

Graduate College
The University of Iowa
Iowa City, Iowa

CERTIFICATE OF APPROVAL

MASTER'S THESIS

This is to certify that the Master's thesis of

Satpreet Harcharan Singh

has been approved by the Examining Committee for the thesis requirement for the Master of Science degree in Electrical and Computer Engineering at the December 2010 graduation.

Thesis Committee: ____________________
Anton Kruger, Thesis Supervisor

____
Jon Kuhl

____
David Bennett

ACKNOWLEDGEMENTS

First, I would like to thank Prof. Anton Kruger, who has been my supervisor, mentor and thesis-adviser through two semesters of Teaching Assistantships, and through five semesters of part-time and full-time work at the IIHR. It was due to his encouragement and advice that I took on a then unfamiliar domain to do my MS Thesis in. To say that I have learned a lot from him in the past two years would be an understatement.

I would like to thank Prof Witold Krajewski, Director - Iowa Flood Center, for his support and guidance throughout my stint at the IIHR.

I am grateful to the team at the IIHR: To Charlie Gunyon, whose assistance during the handover period was extremely helpful and for his continued support to the Hydro-NEXRAD project; To Bongchul Seo and Piotr Domaszczynski whose work with hydrometeorological algorithms, modules and radar-data was integral to the project; To Radek Goska, for developing the GUI for the the Hydro-NEXRAD projects, and for being an affable person to share an office with; To Ricardo Mantilla, for being the first 'customer' for our real-time data and providing us valuable feedback; To Brian Miller (and Mark Wilson) for their indispensable system-administration support.

I would also like to thank our collaborators: From Unidata, Boulder, CO: Jeff Weber and Jeff McWirther amongst others at Unidata, for their ongoing help in improving our setup for receiving NEXRAD radar-data in real-time; From University of Illinois, Urbana-Champaign: Sanjeev Jha and Prof. Barbara Minsker's group, for helping us test a feed for the system.

I appreciate the feedback received, during progress-review presentations, from the other members of my thesis committee: Prof. Jon Kuhl (Electrical Engineering) and Prof. David Bennett (Geography).

I am grateful for the unflinching support of my family back in Pune, India, and to my uncle, Jasbir Arora, and his family here in Iowa, throughout my time in graduate-school.

Last but not the least, I would like to thank the following friends for their encouragement and for the diversions that were so essential to making it through this challenging experience: Sameer, Nick, Nathan, Bhavna, Dmitry, Alejandro, Stacey, Maciej, Aneta, Pranav, Rajat, Siddharth K., Arjun, Siddharth D., Nachiket, Jasmina, Rose and Misbah.

This project is supported by NSF grant 1-52115-00 (NSF 839576).

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure

CHAPTER 1
INTRODUCTION

Hydrometeorology researchers use data-products like rainfall maps, quarter-hourly, hourly and daily rain accumulations, and reflectivity maps to drive their hydrometeorology models.

There are many sources of data available from which to generate such products, e.g. (1) Weather radar reflectivity scan data, (2) Rain-gages and (3) Satellite maps, to name a few.

However, hydrometeorology researchers prefer to work directly with the data-product, rather than deal with the details of sensor data-collection, data-management, storage, quality-control, processing and format conversion.

The Hydro-NEXRAD system (henceforth referred to as HNX1) developed at IIHR-Hydroscience and Engineering at The University of Iowa, provides rapid and user-friendly access to such user customized hydrometeorological products, generated using archived Weather Surveillance Doppler Radar (WSR-88D) data from the NEXRAD radar-network [1] in formats convenient to researchers. HNX1 is a batch-processing system in that users are expected to log-on to the system, set up a customized processing job, submit it and then wait until the system informs them by email that their job is complete, and then collect the job's output via FTP.

Many researchers would like a real-time continuous 'feed' of customized hydrological data-products to drive their hydrological models in a real-time continuous fashion. A new system, Hydro-NEXRAD II (henceforth referred to as HNX2), is currently being developed at IIHR-Hydroscience and Engineering to address this need. Figure 1 shows a high-level overview of the system.

HNX2 differs from a government or mass-media weather map display in that:

- users can receive data in a relatively much higher resolution (explained in section 2.2). HNX2 data-products are generated from the highest resolution of radar-

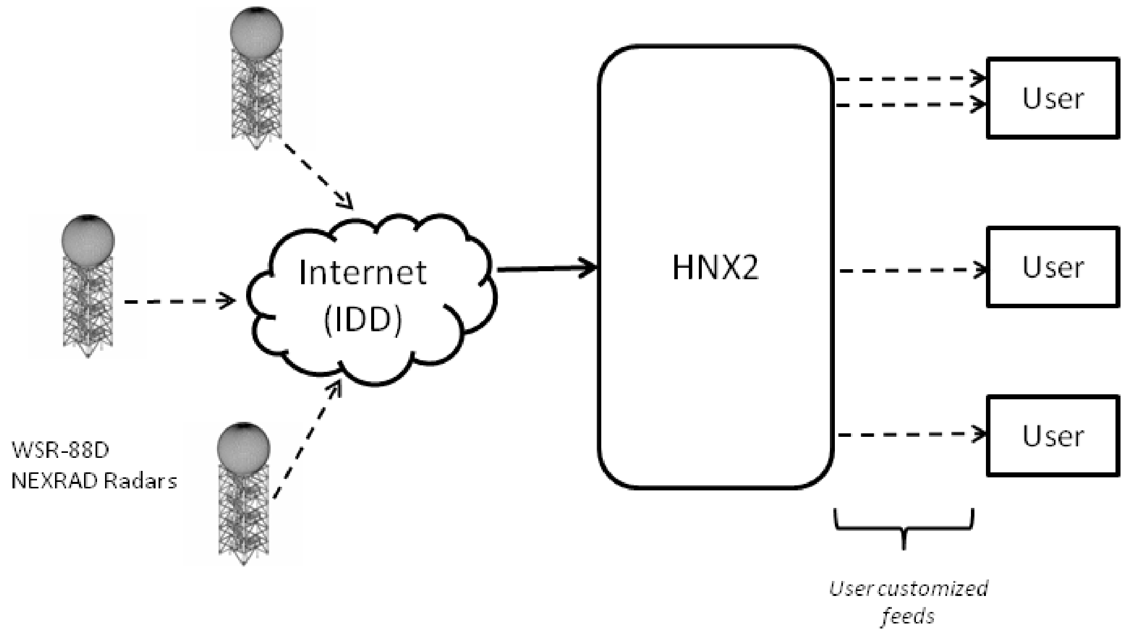


Figure 1.1: The diagram shows a high level overview of the HNX2 system: near real-time Radar data is received at the system at regular intervals from NEXRAD radar sites through an Internet based delivery system called IDD (Internet Data Distribution). This data is then processed according to user preferences for the feeds that have been set up and is available in near-real time to the user.

data (Level II data) available.

- users can restrict the spatial-domain of the data to only the hydrological-basin of interest. As explained in section 2.2, the National Weather Service provides Level III data feeds which are spatially restricted to specific radar-coverage areas but not to hydrological-basins. This is an important feature because hydrology research is primarily concerned with basin-centric research, and is unconcerned with radar coverage-areas.

- users can conveniently customize the algorithm used for processing the input radar-data, and use state-of-the art algorithms developed at IIHR. No customization is possible for a commercial source (e.g. Weather.com [2]).

- users can instrument their hydrological models (which use this data) to programmatically check for and download new data as it becomes available. For a commercial source (e.g. Weather.com [2]), one would have to rely on tedious screen-scraping to programmatically avail the data.

- feeds from HNX2, on the other hand, usually have a higher latency (delay) than from other sources because of the inherent latency (see section 2.2) of the higher-resolution radar-data being received and processed by the system.

The architecture, design and engineering details of HNX2 are the subject of this thesis.

CHAPTER 2
BACKGROUND

This chapter describes the important tools and technologies that were used in the development of HNX2. The HNX1 system has also been described.

2.1 NEXRAD

The National Weather Service (NWS), which is an agency of the National Oceanic and Atmospheric Administration (NOAA) within the US Government's Department of Commerce has been operating a network of weather-radars since 1988. This radar-network, called NEXRAD (NEXt-generation RADar), consists of 159 radars covering the Continental US (see figure 2.1) and other outlying regions.

Individual radars are often referred to as WSR-88D: Weather Surveillance Radar, 1988, Doppler. Four-letter call-signs (e.g. 'KDMX' for Des Moines, 'KDVN' for Davenport) are used refer to individual WSR-88D radars. NEXRAD was originally set up to track severe weather phenomena and according to the Radar Operations Center website [4], "The NEXRAD network provides significant improvements in severe weather and flash flood warnings, air traffic safety, flow control for air traffic, resource protection at military bases, and management of water, agriculture, forest, and snow removal."

Each radar's scan covers a region ("radar-umbrella") of a radius of 250 nautical miles [4]. Radars operate in primarily two scanning modes: Clear-air mode and Precipitation mode [5], encoded in terms of Volume Coverage Patterns (VCPs). In Clear-air mode, a scan is made approximately once every 10-minutes, and in Precipitation mode approximately every 4-5 minutes. The reader is referred to tutorials from NOAA [6] and other sources [7] for more details on how NEXRAD Doppler weather-radars work

NEXRAD radars upgraded the format of their data around mid-2008 from what was referred to as legacy-resolution (or normal-resolution) to a so called super-resolution

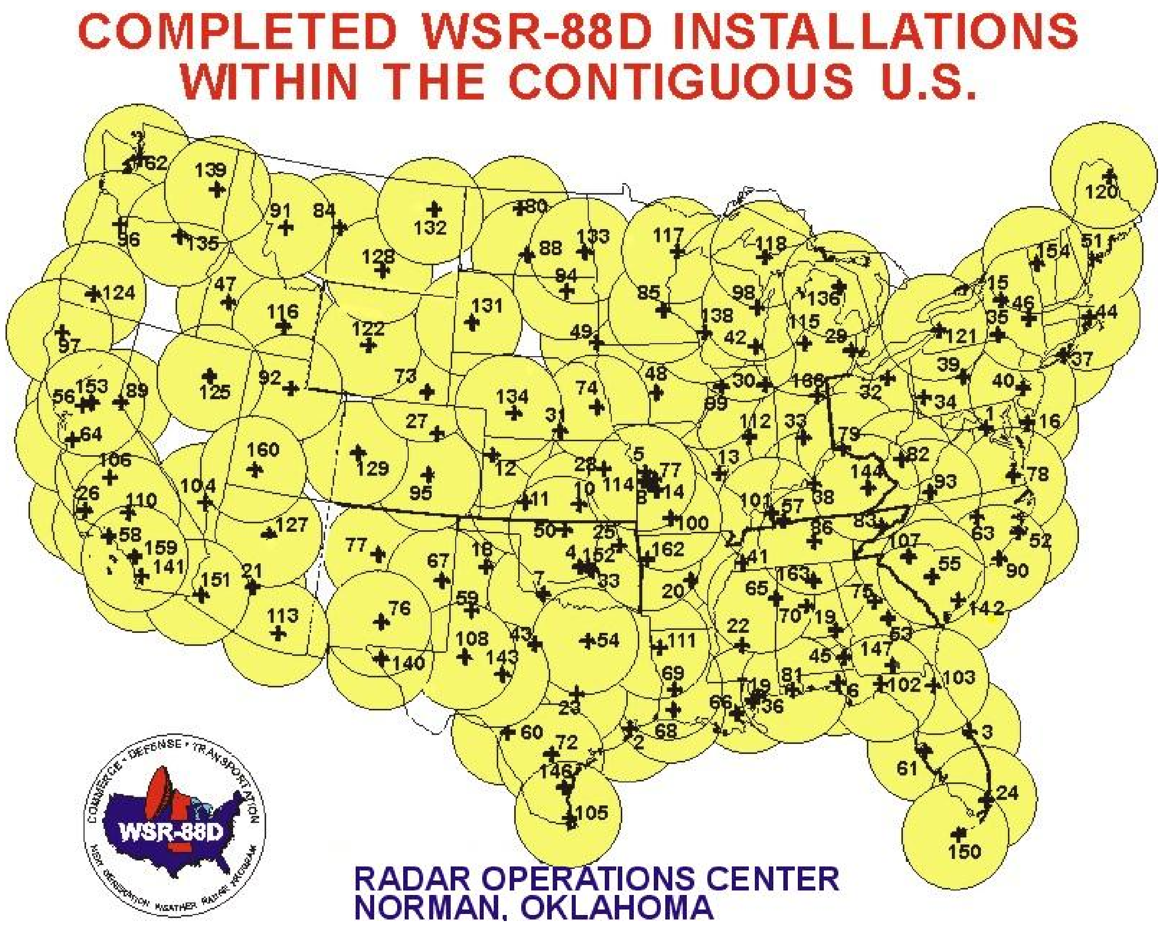

COMPLETED WSR-88D INSTALLATIONS
WITHIN THE CONTIGUOUS U.S.
RADAR OPERATIONS CENTER
NORMAN, OKLAHOMA
WSR-88D
COMMERCE · DEFENSE · TRANSPORTATION
NEXT GENERATION WEATHER RADAR PROGRAM

Figure 2.1: NEXRAD Radar Coverage within the Continental United States: This image has been taken from the website of the National Weather Service's Radar Operation Center, Norman, Oklakhoma [3].

format.

2.2 Radar Data Formats

NEXRAD data received through LDM are classified into two different types. They differ in format, resolution and type of data-product.

2.2.1 Level II data

Level II data, also referred to as 'base' data, are NEXRAD data in the original resolution as captured by the radar. Each file consists of three meteorological data quantities: (1) reflectivity, (2) mean radial velocity, and (3) spectrum width. [8] The number of scans and the frequency of the production of the data depends on the specific scanning strategy being used at that time at the radar. However, in general, each file contains scans in conical-planes (of a spherical-polar coordinate system) at many 'elevations', each of which contains rays at different azimuths covering one full rotation (360 degrees). The azimuthal resolution and radial resolution of rays is 0.5-degree and 0.25 km respectively in the super-resolution format for reflectivity data [9].

Each Level II data file consists of a volume-scan header followed by data and message chunks, each with it's own header. The volume-scan header is 24 bytes and includes data such as the call-sign, latitude, longitude and elevation of the radar at which the file was generated, time at which the scan started and other miscellaneous metadata. Level II data need preprocessing before use in our system because besides useful reflectivity-data they also contain "status messages, performance/maintenance data, volume scan strategy, clutter filter bypass map, and wide-band communication console messages" [10]. In HNX2 we are only concerned with reflectivity data.

2.2.2 Level III data

Level III data are "radar product data generated from the Level II base data to assist forecasters in weather analysis, predictions and warnings." [11]. They consist of reduced-resolution preprocessed data for a variety of different type of products: reflectivity, velocity, storm tracking information, hourly precipitation totals and radar status logs etc. [12]. As these data are of a lower resolution, we do not use them in HNX2.

2.3 Radar Data Reception

The Unidata Program Center [13] of the University Corporation for Atmospheric Research (UCAR) has been distributing geosciences data using their Local Data Manager (LDM) [14] software in near real-time over their Internet Data Distribution (IDD) [15] system since 1994 [15].

2.3.1 Internet Data Distribution (IDD)

The IDD system is a cooperative hierarchical tree-topology network, with Unidata servers at the root of the tree and servers from the over 130 [15] participating universities and institutions as other nodes. The IDD system also allows any participating site to inject it's own dataset into the IDD for delivery to other interested sites. The IDD essentially establishes a overlay-network over the Internet with each node relaying up-stream data to down-stream nodes, in a hierarchical fan-out topology.

Given the dependence of the system on participating nodes, there are inherent uncertainties on uptime and network-delays associated with the system. However, as noted on the Unidata website, they "have been fortunate to have enough sites with adequate resources to act as relay nodes" [15]. No up-time guarantees are provided for the service. To maximize reception up-time, we receive our data directly from Unidata which is at the root of the IDD hierarchical tree-network for distributing NEXRAD data to institutions.

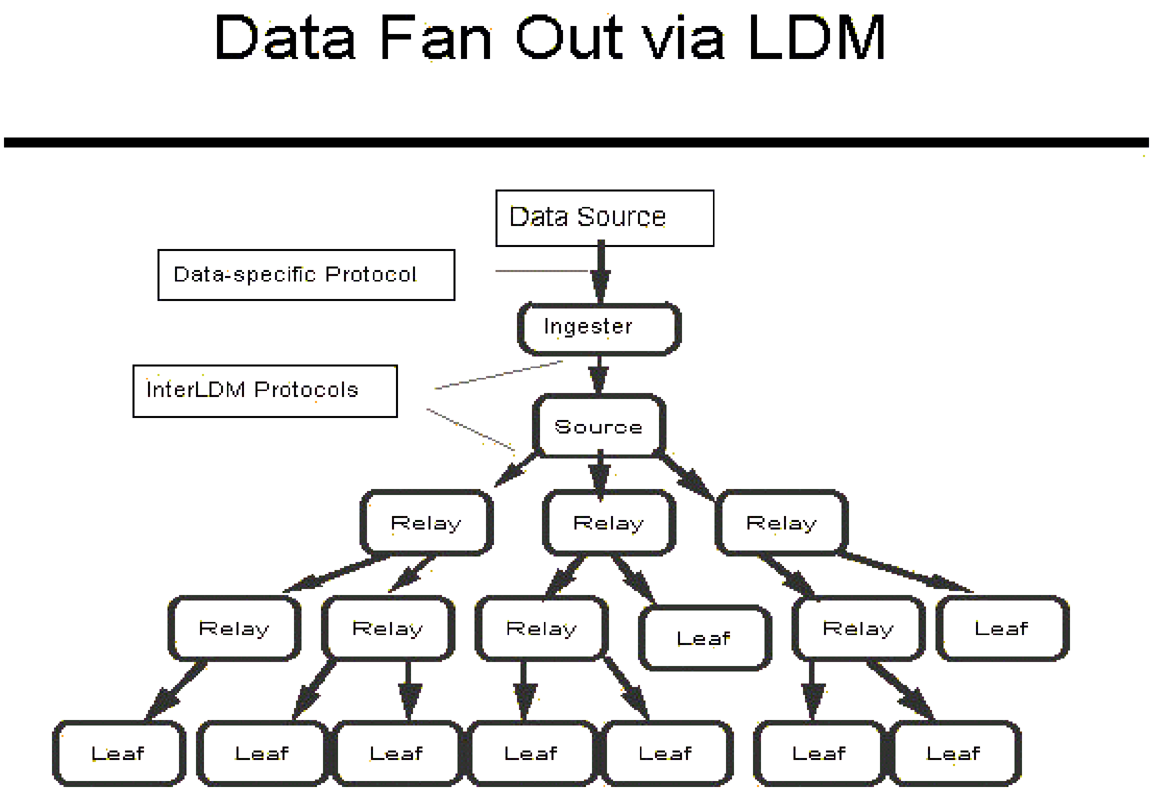


Figure 2.2: Hierarchical fan-out topology of the IDD: The image shows the hierarchical fan-out topology of the Internet Data Distribution. Down-stream nodes are dependent on Up-stream nodes for receiving data. For the purpose of receiving NEXRAD Level II radar-data, the Unidata Program Center is at the root of the tree. This image has been taken from the website of the Unidata Program Center, Boulder, Colorado [16].

2.3.2 Local Data Manager (LDM)

The LDM is a "collection of cooperating programs that select, capture, manage, and distribute arbitrary data products." [15] LDM uses the IDD network-infrastructure to "deliver an aggregate total of about 50 gigabytes of data per day" [15] to participating institutions. The use of LDM to distribute data is not dependent on the IDD network. LDM can be configured to form a distribution overlay-network over any standard TCP/IP network.

Data is pushed on to LDM from its source node with some identification metadata to help identify it from other data flowing on the network. A simple regular-expression match [17] is usually used to decide what data is to be 'pulled' from the parent node. To distribute data to receivers other than those below the position of the source on the hierarchy, data needs to be first transferred to a node higher on the hierarchy, whose sub-tree encompasses all nodes that need to receive the data.

In our system, we configure our LDM installations to receive NEXRAD Level II data from specific radars that the system is currently configured to serve. Due to the continuous streaming nature of the received data, the I/O overhead of each running LDM instance is significant. Due to the large size of the individual files, Level II data is distributed exclusively through IDD/LDM.

2.4 NCDC

The National Climatic Data Center NCDC is the world's largest active archive of weather data [18], and an alternative source of NEXRAD radar data.

However, data from NCDC is not available in real-time and thus not used directly in our system. The NCDC archives have NEXRAD data available with a 24-hour delay from the time they are produced. The NEXRAD Inventory Search [19] tool allows the public to obtain NEXRAD data (24-hours of data at a time for a single site) from the NCDC's archives, which store data for all NEXRAD radars from the start of 1991 to 24-hours from current-time.

We use NEXRAD data from the NCDC archives from time to time, for testing and verification purposes.

2.5 RSL Library

The NCDC website lists many tools that may be used for decoding and reading NEXRAD radar data. [20]

At the IIHR, we use the Radar Software Library [21] which is an object-oriented C-language library developed by the NASA TRMM Satellite Validation Office, in support of their Tropical Rainfall Measuring Mission's (TRMM) Ground Validation Program [22].

The library reads Level II data into a data-structure/object in memory. It provides fine-grained access to scans, with the ability to address data within individual sweeps (at varying elevations), rays (at varying azimuths) and bins (at varying radial-distance).

2.6 Modules

The core task of the HNX2 system is to convert radar data to data-products that are customized by and usable by hydrometeorology researchers. This conversion requires the application of hydrometeorology algorithms that convert radar-reflectivity data to rainfall-maps and accumulation-maps for the area under study. The details of these algorithms are beyond the scope of this manuscript, but for further information the reader is referred to papers by IIHR researchers on radar-rainfall products and associated algorithms [23] [24].

Two PhD students at the IIHR, Bongchul Seo and Piotr Domaszczynski, worked on the development of radar hydrometeorology algorithms for HNX2. The development process established at the IIHR dictated that they provide their work in the well-encapsulated form of a C-program, called a Module, which was executable on any standard Linux machine.

Therefore, for the purpose of this discussion, Modules are treated as black-boxes that have certain preconditions that had to be met in order for them to carry out a well-defined algorithmic operation, and certain postconditions that had to be taken into consideration to use their outputs effectively.

The complete process of conversion of radar-data into a user-customized data product is a complex process consisting of a sequential pipeline of Modules. For the array of options available in HNX2, there actually exist 3 distinct processing pipelines as shown in the highly simplified diagram in figure 2.6:

Brief descriptions of the Modules follow:

For the Single-radar pipeline:

- SPRE: Preprocessing module that takes in quality-checked Level II radar files, and extracts reflectivity data for the selected radar coverage area.
- RATE: Converts reflectivity data to 5-minute rain-rate maps using a preset or user-customized hydrometeorology algorithm. The following keywords are used to refer to some popular algorithms associated with hydrometeorology data: Quicklook, Hi-fi and Pseudo-PPS. For research purposes, the user is also allowed to customize the algorithmic parameters used for processing. The reader is referred to the following papers for details on the algorithms: [23] [24].
- ACCU: Accumulates multiple rain-rate maps into a single 15-minute or 60-minute rainfall-accumulation map.

For the (Multiple-radar) Product-based merging pipeline:

- MPRE: Preprocessing module that takes in quality-checked Level II radar files and extracts reflectivity data for the basin under study.
- MRATE: Converts reflectivity data to 5-minute rain-rate maps using a preset or user-customized hydrometeorology algorithm.

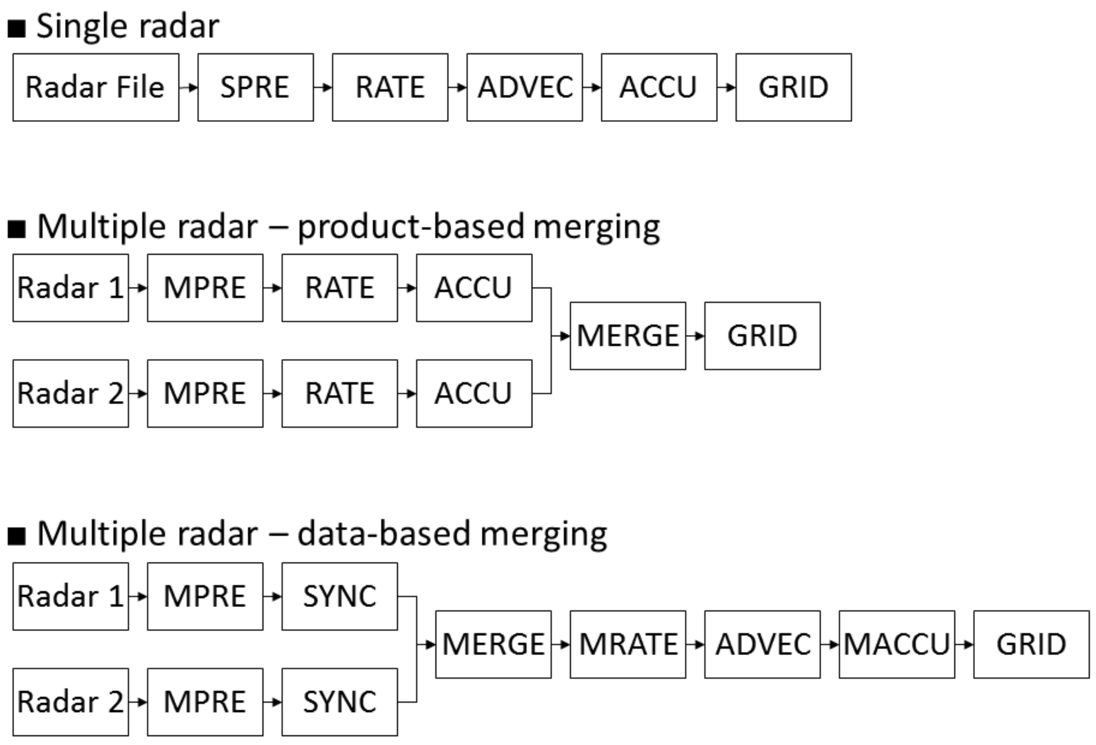


Figure 2.3: HNX2 Processing Pipelines:The diagram shows the various Modules in HNX2. Based on the processing options available to the user, the Modules are divided into three sequences called processing-pipelines. The single-radar pipeline is the simplest and produces only radar-centric products viz. products that cover a radar's coverage-umbrella area. The two merging pipelines allow for the generation of basin-centric products. Basin-centric processing usually requires data from two or more radars to be merged to generate products that cover the basin-area. In HNX2, two merging styles, product-based and data-based, are available based on the choice of products and algorithms [23]. The diagram above is a highly simplified representation and the actual HNX2 pipelines are more complicated with many more branches. The complexity of the pipeline, however, does not affect the design of HNX2 in any significant way.

- MACCU: Accumulates multiple rain-rate maps into a single 15-minute or 60-minute rainfall-accumulation map.

- MERGE: Takes files, for data with the same time-stamp, from multiple radars and generates a single file for the basin-under study. Handles conditions like overlapping data and missing radars.

Additionally, for the (Multiple-radar) Data-based merging pipeline:

- SYNC: Takes adjacent reflectivity-data files and produces an interpolated file for a nominal (synchronized) time-stamp.

Common modules:

- GRID: Converts the final data-file to a standardized grid based on the user's selection. The following are some popular grids associated with hydrometeorology data: ldap, hrap, shrap, latlon, polar. [24]

- FORMAT: Converts the final data-file to a well known format based on the user's choice. The following are some popular formats for data exchange in the Geosciences research community: ascii, arcascii. [24]

2.7 Previous work: HNX1

HNX1 is the predecessor to HNX2. HNX1 allows for user-friendly access to user-customized Geosciences data-products viz. reflectivity maps, rain-rate maps and 15-minute, 60-minute and daily accumulations, from *archived* NEXRAD radar-data.

As depicted in figure 2.7, a user logs into his (or her) HNX1 account using a web-based GUI that can be accessed from any browser to avail the data. The user is required to select temporal (time-span for which products are to be generated) and spatial constraints (basin/radar-coverage area), and parameters (algorithms, map projections / grid, output format) for the generation of the data-products he seeks. HNX1 is a job-driven system, where each request for user-configured products, as

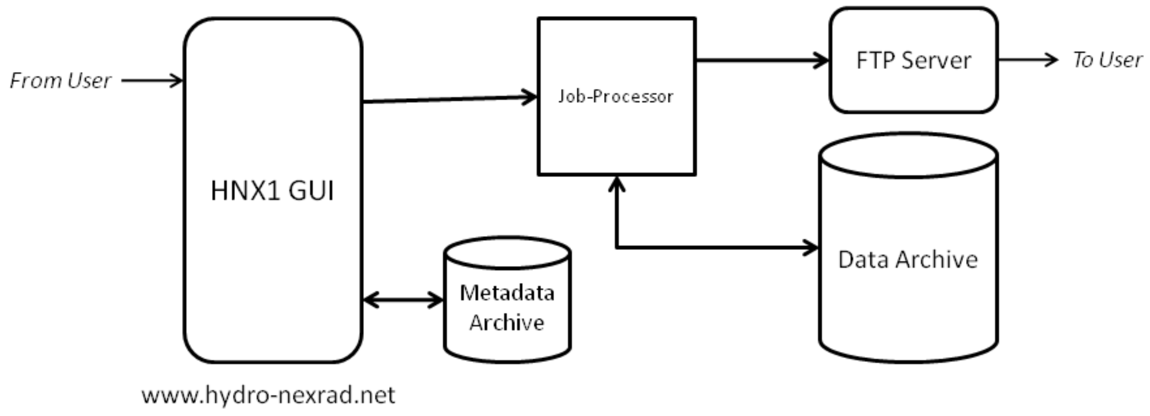


Figure 2.4: High-level overview of HNX1: The GUI allows the user to select time-spans of interest by offering searching over metadata stored in the Metadata Archive. This allows for a much speedier set-up of a processing job than querying the Data Archive itself. Once set-up, the job-processor fetches the relevant data from the Data Archive and carries out relevant processing. When done, it loads the generated products onto an FTP server and notifies the user by email.

entered through the GUI, becomes a job. When the job is complete, a notification is sent to the user with an FTP location from where he can download the generated products.

HNX1 has been in operation since 2006 and currently has over 100 users. User-configured products can be generated from archived data for about 40 NEXRAD radars, with data available till mid-2008 going back about 7 years.

The HNX1 system has a significant maintenance overhead. The archived radar-data in its file-servers total to about 17 TB and it's database is over 250 GB. The database and archive were 'sealed' when the NEXRAD radar-network switched from their legacy data format to the so called 'super-resolution' format around mid-2008.

From the description above, one can conclude that HNX1 is an ideally suited for studying specific historical weather-events or case-studies. It is not well suited to a situation where a current or ongoing event needs to be studied. HNX2, which provides user-customized data in the form of real-time feeds, is designed for such applications.

For more details on HNX1, the reader is referred to papers that deal with efficient storage and fast read-time data-format [25], relational database for flexible data organization [26], metadata calculations [27] and radar-rainfall products and associated algorithms [23] [24].

2.8 Near Real-Time

In this manuscript, the term Near Real-Time has been used to contrast HNX2 with the batch-processing methodology of HNX1. In computing literature, the term *Real-Time* is generally used to imply that a system guarantees a certain action to take place after a trigger (or event) by a certain time-deadline. Additionally, the terms *Hard* and *Soft* Real-Time are used to identify systems where the consequence of taking an action later than it's required deadline *has* or *doesn't have* (respectively) catastrophic consequences [28]. HNX2 is a soft real-time system in that it produces data at regular deadlines (see section 3.1 for details), with slight variations (in the order of seconds) in the exact time that the data is available to the user (depending on system load).

However, the term *Near Real-Time* is more suited to streaming data systems such as HNX2. Our usage of the term "near real-time" is compliant with the US Department of Defense's Dictionary of Military and Associated Terms definition: "Pertaining to the timeliness of data or information which has been delayed by the time required for electronic communication and automatic data processing. This implies that there are no significant delays" [29]. In HNX2, the primary sources of delays are: the time taken for the transport of data from the radars involved in generating a feed to the HNX2 servers, and the time required to collect, process and publish the received data.

CHAPTER 3 PROBLEM STATEMENT

3.1 Desiderata

Based on discussions with stakeholders, the following desirable attributes were identified for the Hydro-NEXRAD 2 system:

- **Near Real-time**: HNX2 should deliver it's data products in Near Real-Time (as described in section 2.8).
- **Products**: At the least, the system should provide the following products:
  - Rain-rate Maps: Every 5-minutes (aligned to clock. eg. 10:00. 10:05, 10:10 ...)
  - Quarter-hourly Accumulations: Every 15-minutes (aligned to clock. eg. 10:00. 10:15, 10:30, ...)
  - Hourly Accumulations: Every 60-minutes (aligned to clock. eg. 10:00. 11:00, 12:00, ...)
- **Usable**: The system must be easily usable by researchers in the Geosciences, without any requirements of extensive computer knowledge or system administration skills. The user does not have to deal with the details of sensor data-collection, data management, quality-control, processing and format conversion.
- **Platform Independence**: It is important that at the least, Windows and Unix-like systems should be able to use the HNX2 system. Ideally, the system should be platform independent and work within a browser.
- **Authentication**: Users should have individual accounts, which they have to log into to view their active feeds.
- **Scalable**: The system must be able to process multiple parallel feeds and must be able to grow to handle data from a large number of NEXRAD radars.

- **Inexpensive**: The system should use open-source software to reduce costs. It must require minimal administration in terms of initial setup and ongoing costs.
- **Administrative functions**: Some of the important administrative features identified include:
  - The ability to add or remove users in bulk or programmatically
  - The ability to limit job sizes, so that no single user can capture more than his fair share of resources in the system.
  - The ability to delete old files automatically, to make space for newly generated files from the real-time feed.
- **Data-delivery**: A user should be able to manually download data from the feeds that he has set up, or programmatically using a script or some other form of automation. The set up process itself should be a one-time task after which the system should behave like a web-service.

3.2 Constraints

The following engineering and infrastructure constraints need to be kept in mind:

- Prof. Witold Krajewski, Director of the Iowa Flood Center, was the primary 'customer' of the project. He would advocate for the requirements of the users of the system from the general hydro-meteorological research community.
- Due to the change of the format of data received from NEXRAD radars, many modules that were written for HNX1 had to be rewritten for HNX2. It was therefore required that an iterative, prototype-driven development process be followed. It was expected that the processing-pipeline would change over the course of the project, and it indeed did, significantly.

- As development was being carried out, it was important that test-feeds be set up to gracefully handle exceptional conditions and corner-cases that might have been missed during the development process.
- LDM was chosen as the source of NEXRAD radar-data from the outset. They are the only known free distribution channel of NEXRAD Level II data to academic institutions.
- We had two test-users for the system. Feedback from these users was incorporated from time to time in the development of HNX2:
  - The Iowa Flood Center, housed at the IIHR, who used real-time 5-minute rain-rate maps generated from seven radars covering the State of Iowa (Call signs: KDVN, KDMX, KMPX, KARX, KOAX, KFSD, KEAX) to feed into flood-forecasting CUENCAS models [30].
  - Prof. Barbara Minsker's group at UIUC, who used real-time 5-minute rain-rate maps generated from three-radars (Call Signs: KLOT, KIND, KILX) covering the Upper Embarras Watershed in Illinois for a 'virtual-sensor' project [31].
- It is important to note that the Web-based GUI was subject to significant reuse from HNX2's predecessor HNX1. The primary developer for the HNX2 GUI would be Radek Goska (a full-time IIHR employee specializing in client-side and graphical UI development), who also worked on the equivalent GUI for HNX1.
- Some logic was left on the GUI end due to the fact that a full-time employee (Radek Goska) would be available for making minor tweaks, as is natural and expected in a research-environment. The HNX2 GUI might also be shared with future projects.

- As the GUI for HNX2 may be shared with HNX1 and other future projects, a very clear boundary and separation of concerns was required between the GUI and other parts of HNX2.

- For testing, it should be possible to set up feeds manually i.e. without the need of a GUI-based setup interface.

- It was important to use same or similar technologies as HNX1 so that minimal training overhead would be required for new personnel. Maximum code reuse and use of open-source tools was recommended.

- The computing infrastructure at the IIHR would have to be used. Enterprise grade servers were available with administration support during standard business-hours during the week. HNX2 code would have to be reasonably robust to recover from upsets like power-outages or momentary resets.

CHAPTER 4
SYSTEM OVERVIEW AND ARCHITECTURE

Essentially, HNX2 is a system which collects near real-time 'raw' data from multiple NEXRAD radar sources, processes them through a pipeline of data-processing modules (which are configured on the basis of the user's requests) and then publishes the processed product-data in a way that is convenient for the user to collect it. In many ways, HNX2 is similar to the HNX1 system, with the major differences being that no metadata-assisted search is possible (because no archival radar-data is stored in the system), and that data products are generated continuously in near real-time.

The discussion of the architecture of HNX2 has been divided into two sections: Frontend and Backend. Figure 4 shows a simplified-view of the architecture of HNX2.

4.1 Frontend

The term Frontend refers to all the end-user facing components of the HNX2 system. This consists of the interface with which a user sets up new feeds or stops new feeds, and the interface used to access data from feeds that are already set up.

The HNX2 GUI is accessible through any standard browser, and will allow the user to log in and configure the parameters of a new product feed. On the basis of the user's entries, the GUI will generate a feed description and pass that on to the back-end through an XML-RPC call.

A 'feed' will be available at a web-accessible URL as provided by the GUI during the feed setup, and will continue till the time the feed is stopped by the user manually.

4.2 Backend

The HNX2 backend consists of the following parts:

1. **LDM**: this is configured to receive streaming data from the various radars for which data is currently required by the system.

2. **Harvesters**: A Harvester takes data from the LDM system, checks it for errors

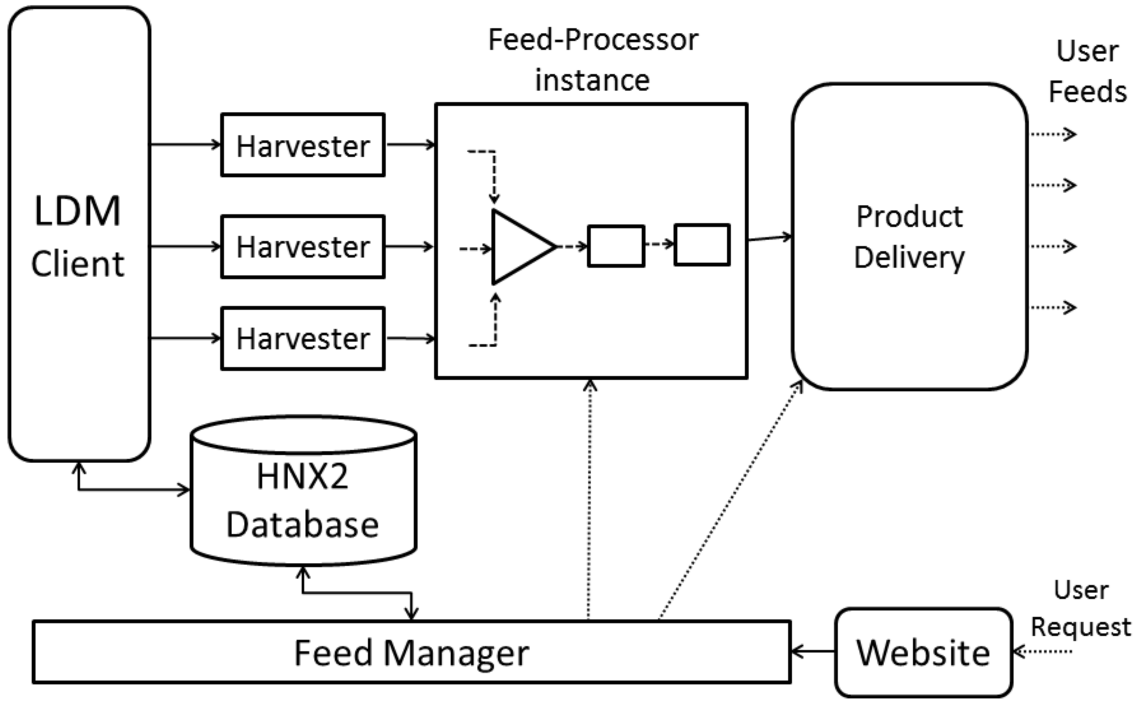


Figure 4.1: Internal structure of HNX2: Users use the web-based GUI to select spatial constraints (basin/radar-coverage area) and parameters (algorithms, grid, output format) to set up a feed. The GUI passes this request on to the Feed-Manager which configures the rest of the system to produce the new feed. The Feed-Processor is invoked at regular intervals and is responsible for the generation of new products. On each invocation, it looks for new radar-files made available to it through the LDM/Harvesters setup, and then passes these files through a pipeline of processing Modules. Generated products are put on a web-accessible Product Delivery interface for the user to collect.

and converts it into a format usable by the next stage, the Feed-Processor. One or more Harvesters provide the data to a single Feed-Processor instance, depending on whether the feed depends on data from single or multiple radars.

3. **Feed-Processor**: the Feed-Processor is the main workhorse of the back-end and runs the harvested data through the pipeline of data-processing modules to produce the data products. This involves time-synchronization of data-streams from multiple radars, rainfall estimation and grid-conversion algorithms similar to those used in HNX1 [24]. The Feed-Processor deposits its data every time a new product is generated to a web-accessible location for the user to obtain the data from.

4. **Feed-Manager**: The Feed-Manager is responsible for the initial setup of the Feed-Processor and for stopping the feed when demanded. Figure 4 shows a simplified-view of the architecture of HNX2 for only one instance of the Feed-Manager and one instance of LDM. As discussed in section 6.5, HNX2 can be scaled to include multiple instances of the Feed-Manager and LDM, all running on separate machines.

CHAPTER 5
SYSTEM DESCRIPTION

HNX2 did not lend itself well to an Object-Oriented Design approach. Instead, a loosely-coupled distributed architecture using many ”Scripts and Pipes” evolved during the development of the system. Well defined interfaces were designed between the larger components of the system.

5.1 Tools and Technologies

Many open-source tools and technologies were used in the development of HNX2:

5.1.1 Linux

Many Linux facilities/features were used instead of writing programs from scratch. Some of these include:

- **cron** - to schedule recurrent tasks
- **scp** (secure copy) - for data transfer between machines/users
- **gzip** - for data compression in a well-recognized format (with open-source libraries and tools)
- **uptime** - for accessing kernel-collected load statistics
- **ntp** (network time protocol) - for keeping the multiple machines in synchrony

The Red Hat Enterprise Linux (RHEL) [32] operating system is used on the Linux servers at the IIHR. Trained system-administration staff manage the servers, and regular security and software updates are available.

5.1.2 Python

Python was used as a scripting and ’glue language’ between utilities. Python was chosen for this purpose for the following reasons:

- It comes with ”batteries-included” i.e. many ready-made modules for an assortment of system tasks.
- It has been used in the IIHR for other projects (e.g. HNX1). Using the same language, helps reduce the learning overhead associated with system maintenance.
- Python is also popular amongst the Geosciences community. E.g. There have been sessions in the annual meeting of the American Geophysical Union on the use of Python in Hydrology [33].

The most important of the Python modules used are:

- **os.system** - for accessing files, system-level services and system-information
- **datetime** - for manipulating datetime objects and performing time-zone conversions
- **subprocess** - for forking algorithm-modules or as a wrapper for other command-line tasks
- **psychopg2** - database connector and query module
- **XMLRPCServer** - for setting up the back-end XML-RPC Server
- **ConfigParser** - for reading and writing configuration files in a human-readable format

5.1.3 Apache

The Apache web-server was used in HNX2 because it has a simple basic-configuration, is open-source, serves all of our requirements, and comes bundled with RHEL.

5.1.4 Postgres

We have used the open-source PostgreSQL server for HNX2, because it has all the features we needed for development, and it has been used in HNX1 and other IIHR projects.

The following sections describe each block of the internal structure of HNX2 as shown in figure 4 and figure 5.4:

5.2 LDM/Harvesting

As mentioned in section 2.3, LDM is our source of real-time streaming data from the NEXRAD radar network.

Harvesting refers to all the steps required to take files received through LDM, discard unusable files, and provide error-free files available to each feed-processing machine and in a format that is readily usable by further stages viz. the processing-pipeline of Modules.

5.2.1 Format Conversion

In the system, LDM is treated as a black-box which produces Level II files at regular intervals for the radars selected for reception. Level II files are read by our internal Modules using the RSL library. However, LDM-received Level II files are not directly readable by the RSL library and require a small format conversion step.

LDM-received Level II files have the following format:

- 24 byte header, consisting of: radar call-sign, latitude, longitude and elevation of the radar, time at which the scan started and other miscellaneous metadata
- multiple chunks of BZip compressed radar data

To make these files readable by the RSL library, a format-converter script was written that carried out the following steps:

- Copy the 24-byte header exactly

- Uncompress each BZip-compressed chunk
- Concatenate all of the above into one file
- GZip compress all of the above

The aforementioned steps were arrived upon after significant experimentation and by reverse engineering Java code in a general Geosciences-data Visualization-Tool [34] on the NCDC website.

5.2.2 Handling erroneous files

It was found that some files received through LDM were incomplete or damaged due to network losses or other random errors. In such cases, the format-converter would fail. As will be described in the section 6.1, this error rate was found to be quite high and a work-around was engineered: An *alternate format-converter script* that converted only the correctly received parts of the LDM Level II file was written. It carried out the same steps on the received file as the original script except that it ignored faulty or incomplete BZip-compressed chunks.

This partially recovered file was checked to be usable by the successive sections of the processing pipeline by a quality-checker program that was developed in consultation with the developers of the modules.

Once a usable file was generated, it was copied (using scp) to the various machines that carry out processing, to predefined locations on their file-system.

The above steps are encapsulated into a script called the *Harvester*. A separate Harvester for each active radar is instantiated every minute by cron. Each Harvester maintains a 'lock file' to ensure that no more than one instance is ever running for the same radar. Each Harvester gets its list of destination hosts from the database.

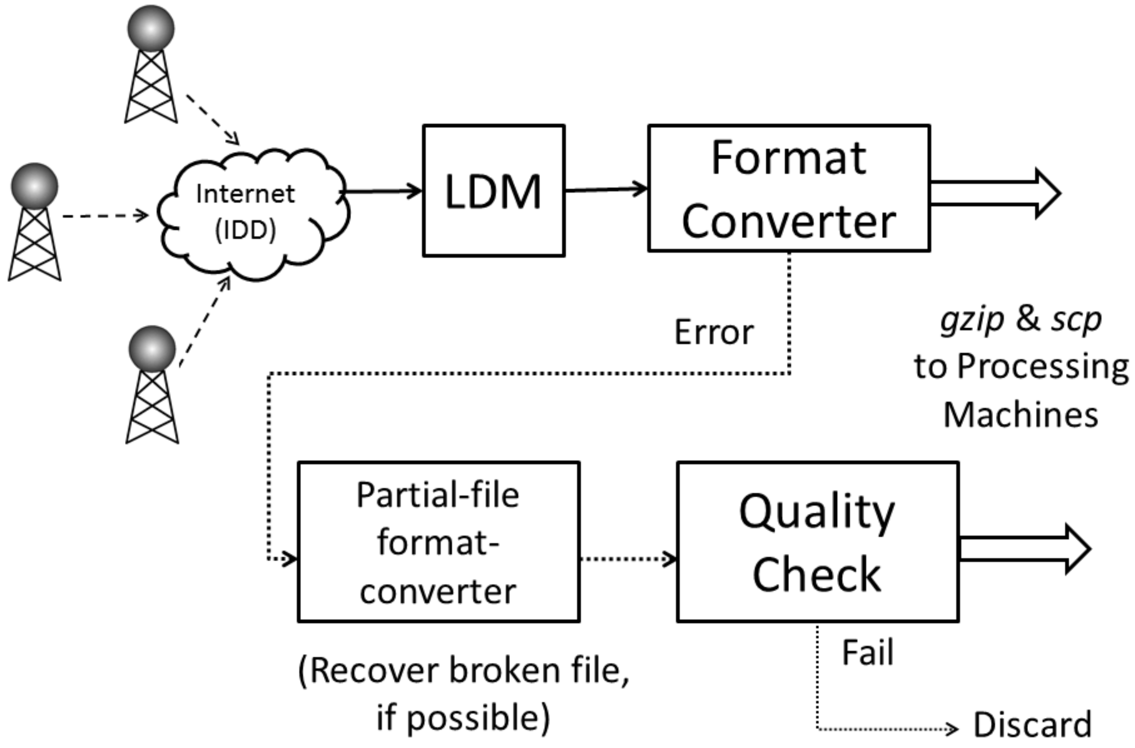


Figure 5.1: Radar-data Harvesting Process: All LDM-received Level II radar files need format-conversion to be readable by the RSL library which is used by the Modules to read and manipulate radar-data. The Format-Converter module carries out this conversion. If a LDM-received file is incomplete, then the Format-Converter fails and a Partial-File Format-Converter script tries to recover whatever radar-data from the recived file as is possible. The Quality-Check script checks if this partial-file is has sufficient data to be usable by the processing-pipeline and discards the format-converted radar file if not usable. Format-converted files, full or partial, are compressed using gzip and transferred to predefined loctions on processing-machines using secure-copy (scp).

5.3 Web-based GUI (End-user interface)

The Web-based GUI provides a user-friendly platform-independent interface that a user can use to interact with HNX2. It consists of:

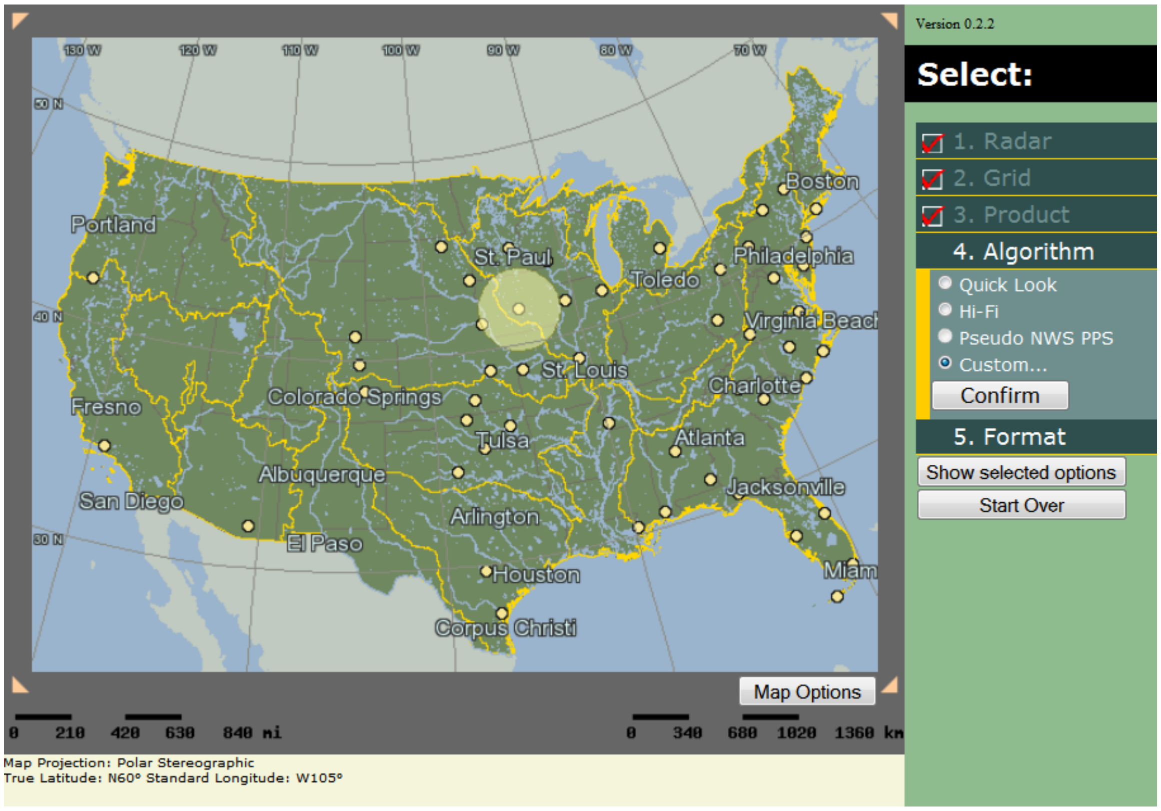
- The user-account and authentication interface
- User-Account and Feed management interface
- New Feed selection interface

There are two types of users in HNX2: (normal-) users and administrators.

Each user has an account-name (their email-address) and a password with which they can log into HNX2. Only administrators can add new or remove existing users. The HNX2 GUI also has a link that allows new users to sign-up for a new account on the system.

On logging-in the user is taken to a page on which links to all his currently running feeds is displayed. The page allows the user to delete any currently running feed or to start new feeds. The new-feed setup page allows a user to setup a radar-centric (single-radar) feed or a basin-centric (usually multiple radars) feed. The user selects the spatial-domain using a convenient map-based interface, the type of product (e.g. rain-rate map, accumulation), the algorithm (preset or custom), the merging paradigm if applicable (product-merging or data-merging) and the output format and grid that he would like to obtain the products in. Figure 5.3 shows a snapshot of the feed-setup interface for algorithm selection.

After selecting the desired options, the user submits the feed.Based on the user's selected options, the GUI sets up a new job on a processing machine by making a Remote Procedure Call (RPC) to it's XML-RPC Server. If successful, the user is returned a URL where the feed's products will start appearing shortly. If unsuccessful, the user is shown an error message along with relevant information.



Map Projection: Polar Stereographic
True Latitude: N60° Standard Longitude: W105°

Figure 5.2: Snapshot of the HNX2 GUI showing the feed-setup interface. Note the convenient map-based interface for selecting spatial-domain (radar or basin).

5.4 Feed-Manager

The Feed-Manager is where the HNX2 'Backend' starts. It is a multi-threaded Python application with an XML-RPC Server running on one independent thread. The XML-RPC Server services requests coming from the GUI and provides procedures for the setup or stopping of new feeds. XML-RPC provides a well defined interface between the GUI and the Backend using a well-known protocol.

For the purpose of scaling the system to serve many feeds simultaneously, HNX2 may have many processing machines set up. As shown in figure 5.4, each processing machine has exactly one instance of a Feed-Manager running on it, and may have multiple instances of Feed-Processors running on it.

Each processing-machine has a local radar-data buffer into which all running Harvesters deposit the latest format-converted and quality-checked radar files. The local radar-data buffer is merely a location (with a self-describing directory structure) on the machine's local file-system dedicated to the purpose of storing usable Level II radar data.

The starting of a new feed involves the following steps:

- Receiving a well-formed XML-RPC request from the GUI for starting a new feed.
- Setting up a web-visible directory where the feed's products will be available
- Gathering relevant radar-centric and basin-centric meta-data (e.g. radar elevations, radar lat-long, basin bounding-box) from the HNX2 database
- Generating a feed-configuration file (named *feed.conf*). This file encapsulates all the configuration-information a Feed-Processor needs to generate a feed's products.
- Inserting one or more entries into the local crontab for cron to schedule the recurrent instantiation of the Feed-Processor

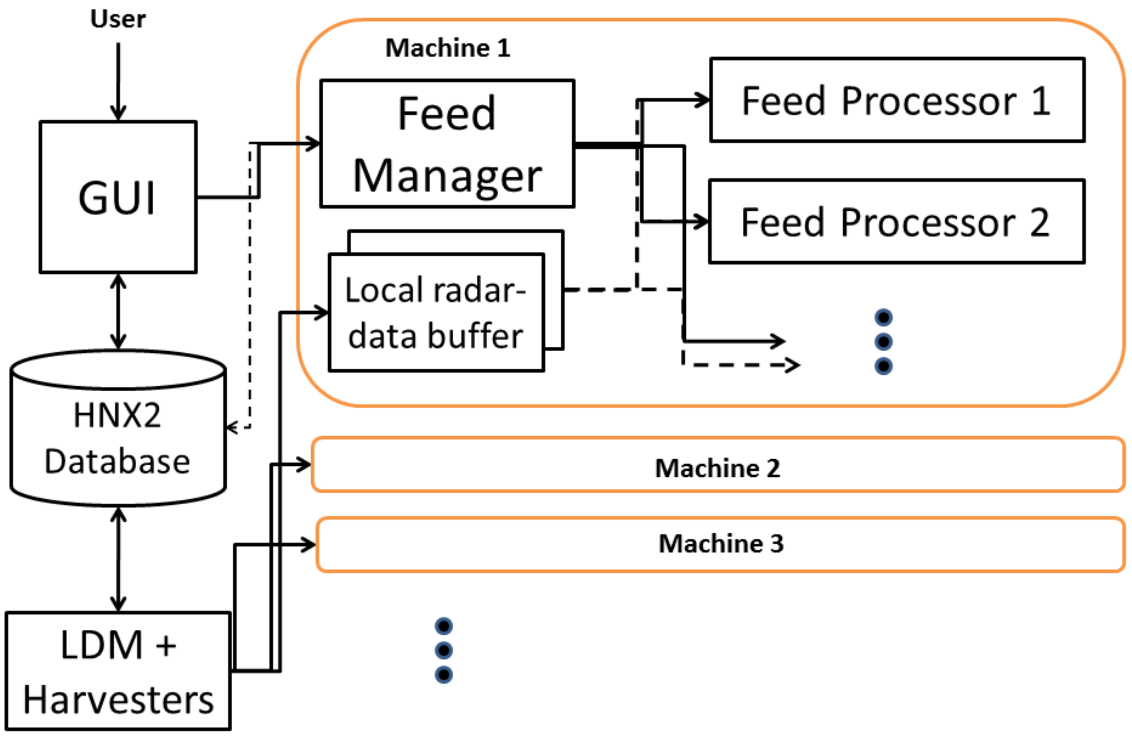


Figure 5.3: Alternative view of the internal structure of HNX2: To be scalable, HNX2 can have multiple processing-machines set up for producing feeds. Each processing-machine has exactly one instance of the Feed-Manager running on it, and can in turn set up multiple instances of Feed-Processors (one for each feed). The Feed-processors in turn instantiate relevant Modules to generate user-customized products. The LDM/Harvesters setup deposits Feed-Processor-usable radar-data on each processing-machines at a predefined location on the file-system referred to as the Local radar-data buffer.

- Inserting an record into the database about the new feed
- Returning a URL corresponding to the directory where products will be available. If any of the above steps fail, returning an error message.

Stopping a feed involves the following steps:

- Receiving a well-formed XML-RPC request from the GUI for stopping an existing feed.
- Checking for the presence of a record into the database for the existing feed. If not available, returning an error message. If available, carry out the following remaining steps.
- Remove the database record, crontab entry and files and directories associated with the feed.
- Return an 'OK' message to the GUI. If any of the above steps fail, returning an error message.

5.5 Feed Processor

As introduced in the previous section, the Feed-Processor is an autonomous component that is repeatedly instantiated (by cron) and is responsible for generating user-customized products. It gets it's configuration information from a configuration-file (feed.conf) generated by the Feed-Manager during setup. Based on this configuration, it finds the latest files available for the radars specified in feed.conf from the local radar-data buffer. It uses these files as inputs to a pipeline of preprocessing, algorithmic and format-conversion modules to generate the a new product.

The primary function performed by the Feed-Processor is the sequencing of data through the pipeline of Modules that generate the user-customized product. Most modules are written in C and have rigid preconditions to be satisfied in order to run. Postconditions vary from module to module.

Some example pre-conditions and post-conditions are as follows:

- Input file name must have a certain format
- Input files must have timestamps that are within a certain time limit of each other, but at least a minimum time apart
- Input file should be a plain-text list of data files in the same directory
- Output file will be generated in current directory. Output file will be streamed to stdout
- One or more output files will be generated. In some cases, no output file will be generated

Python provides a powerful set of tools through the 'subprocess' module (to fork new processes and capture stdout, stdin & stderr), the 'tempfile' module (to make temporary files and temporary processing directories) and the 'shutil' module (to perform high level shell-tasks like removing a directory tree)

5.6 Data-Product Delivery

In the present state of the project, we have a very simplistic setup for the delivery of generated products to the user.

Each processing-machine has an instance of the Apache Web Server running on it, configured to show directory-listings of the web-visible directories under it's document serving directory (called 'DocumentRoot').

In our current methodology, the job-processor producing the new product puts the newly-generated product-file at a predefined location, which is visible in the directory-listing at the URL. E.g.

```
http://s-iihr57.iihr.uiowa.edu/feeds/SR/tmpwChIHp/
```

A file called *latest.txt*, containing the name of the latest product-file, is generated by the Job-Processor every time a new product-file is put into the directory. For debugging purposes, the Job-Processor can be configured to also store files generated in intermediate processing-steps from the processing-pipeline.

5.7 Database

HNX2 has a single database to which the following components connect to:

- **GUI**:
  - to authenticate users into the system
  - to get the list of currently active user accounts and feeds
  - to get the list of processing-machines (where new feeds may be started)
  - to add new-user account information
- **Feed-Manager**:
  - to add new-feed or remove current-feed data
  - to retrieve basin and radar metadata for the generation of feed-configuration files
- **Harvesters**: to get the list of processing-machines (where harvested radar-data should be deposited)
- **Miscellaneous development scripts**: to store and retrieve metrics development related metrics (e.g. latency of received radar data file)

The Database thus has tables for:

- Users
- Feeds

- Processing-Machines
- Metrics
- Radar and Basin metadata

CHAPTER 6
ENGINEERING

This chapter describes the engineering and design decisions made during the development of HNX2:

6.1 Reducing the effect of error-rate in LDM

As mentioned in section 5.2, the Harvester is responsible for taking files received through LDM and depositing them at a predefined radar-data buffer location on the file-system of each feed-processing machine.

At an early stage of the project, it was noticed that our system was often not able to display processed data when the equivalent data was available at the National Weather Service website (eg. for KDVN [35]). This implied that the radar was working and transmitting data but the data was not reaching the end of the HNX2 processing-pipeline. On investigation, we discovered that many files received by our LDM system were incomplete, out-of-sequence or corrupt.

6.1.1 Unidata Script

We took up this issue with the people at Unidata, who are responsible for the delivery of radar-data through LDM. We received a script from Unidata which, working within LDM, would ensure that received files were not out-of-sequence/scrambled by either putting the data back in sequence or creating 'partial' files with only continuous in-sequence data. The fraction of erroneous files after this 'patch' was still unacceptably high (10% -15%) for HNX2:

For feeds that merge data from multiple-radars, an error rate this high at the input-side meant that there would be incomplete data for a much higher percentage of data-products on the output-side. Eg. If products were to be generated by merging data from data-streams from two radars, each stream having error-free data 90% of the time (on average); then the combined output-stream would produce complete/error-free data 90% * 90% = 81% of the time (on average).

6.1.2 Quality Checking and Partial file Recovery

On discussions with Module authors, whose programs use the data, it was discovered that only a part of the complete Level II radar-data file was required:

- Reflectivity Volume was required
- Base sweep (of Reflectivity Volume) was required
- 360 rays (for normal-resolution) or 720 rays (for super-resolution) were required in the base sweep
- Other volumes (mean radial-velocity and spectrum width) were not required

As mentioned in section 5.2, an alternate partial-file format-converter script and quality-checker was written to make use of the aforementioned relaxed constraints. This increased the percentage of files that were usable by the rest of the processing-pipeline to 98.5% (ref. table 6.1.3).

6.1.3 Improvement

To quantify the extent of the improvement, the Harvester scripts were instrumented to collect metrics for each received file.

Figure 6.1.3 shows graphs made from the collected metrics for the Davenport (Call sign: KDVN) radar. This is a concise visualization for monitoring LDM/Harvester status.

Results: As shown in table 6.1.3, about 13.5% files that were previously unusable were made usable by use of the correction scripts.

6.2 Radar Data Synchronization

For data-products that are basin-centric, it normally takes the collective span of two or more radar coverage-umbrellas to fully cover the area of the basin. In such

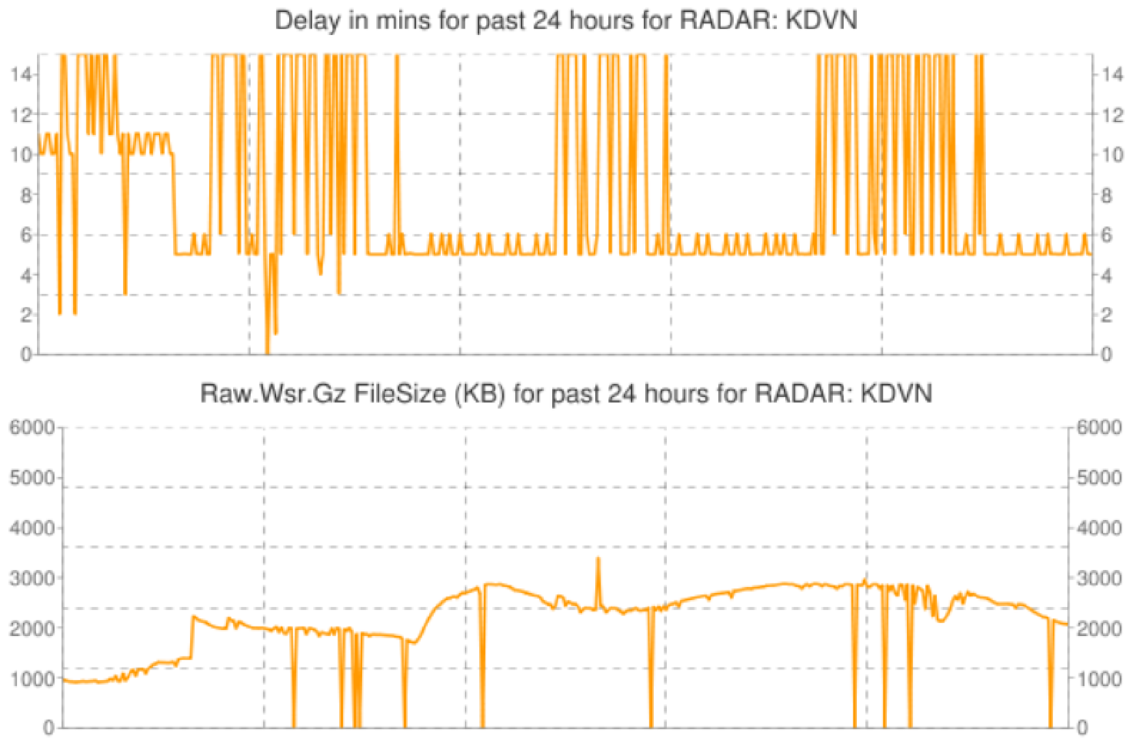


Figure 6.1: Screen-shot showing metrics collected over the past 24 hours for a single radar: KDVN (Davenport). The upper graph shows the variation in delay (in minutes) for files received over the past 24-hours. The lower graph shows the variation in compressed file-size (in KB) over the past 24-hours. The sharp dips in the lower graph indicate instances of received files that were completely unusable.

	Before correction	**After correction**
Usable files	85%	98.5%
Unusable files	15%	1.5%

Table 6.1: Improvement in LDM error-rate on applying correction scripts

cases, data from multiple radars needs to be merged in order to generate a data-product for the entire basin-area.

6.2.1 Need for Synchronization

Radars in the NEXRAD network operate independently and are not synchronized in time. To merge data from multiple radars, it is required that data be available for the same common time-stamp from each radar in consideration. In HNX2, such files are generated by *interpolating* two adjacent (in time) radar data-files for each radar. Once an interpolated-file from each radar is available for the common time-stamp, a merged file is generated by a Module, which can then be passed on to the later stages of the processing-pipeline.

Consider for example the production of 5-minute rain-rate maps in HNX2. Interpolated files that are generated for each radar are aligned with a 5-minute time-boundary. E.g. if radar data files for time-stamps: 10:03 and 10:07 are available, then an interpolated product will be produced for time-stamp: 10:05. Figure 6.2.1 illustrates the interpolation and merging process for multiple radars operating in either scanning mode.

6.2.2 LDM Uncertainties

There are multiple uncertainties associated with the reception of radar-files through LDM:

- Depending on the scanning mode, the frequency with which files are produced varies.
- The network delay associated with each file's transport is variable, as depicted in figure 6.2.2.
- Radars sometimes go off-line for maintenance, and do not transmit any files for a few hours

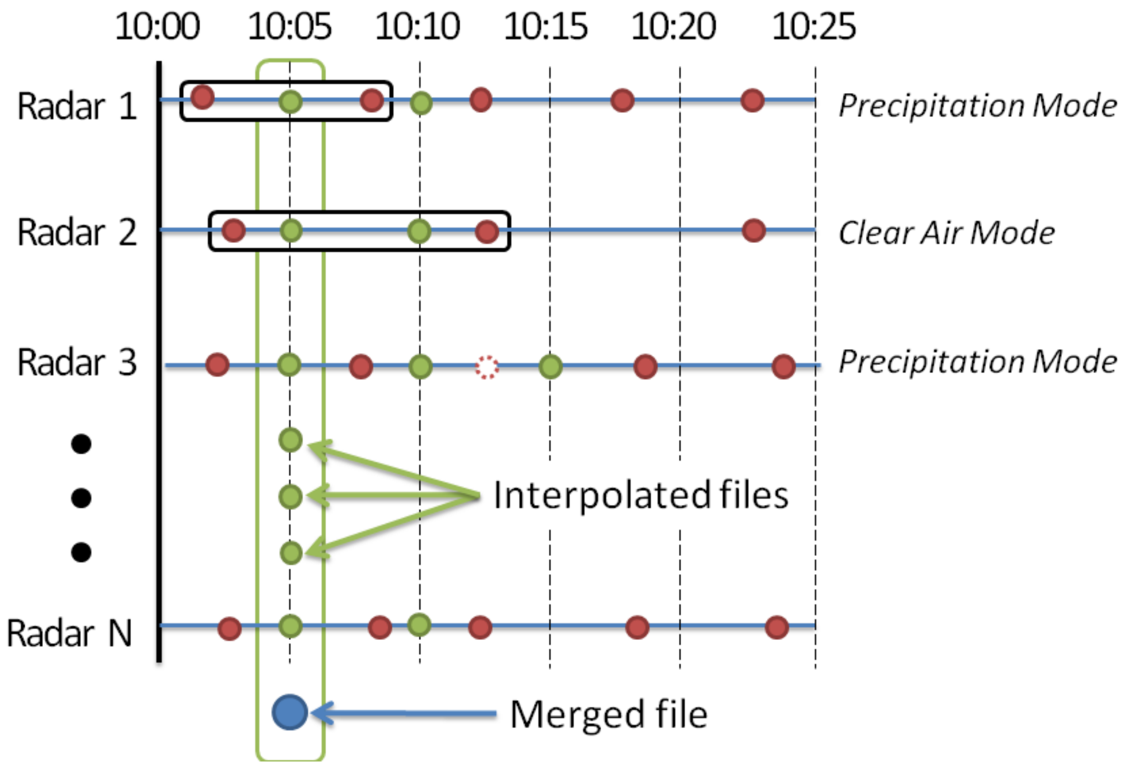


Figure 6.2: Synchronization of data from multiple radars for generation of merged products: In the diagram, horizontal lines have markers representing the times at which radar-data is received from individual radars. For merging, files are generated at nominal-times (:00, :05, :10, ...) by interpolating radar-data files that are adjacent in time (which implies a different waiting period depending on the radar scanning-mode). Nominal-time files for multiple radars can be merged together to form a single merged-file. The data in the merged-file spans the area of the basin under consideration.

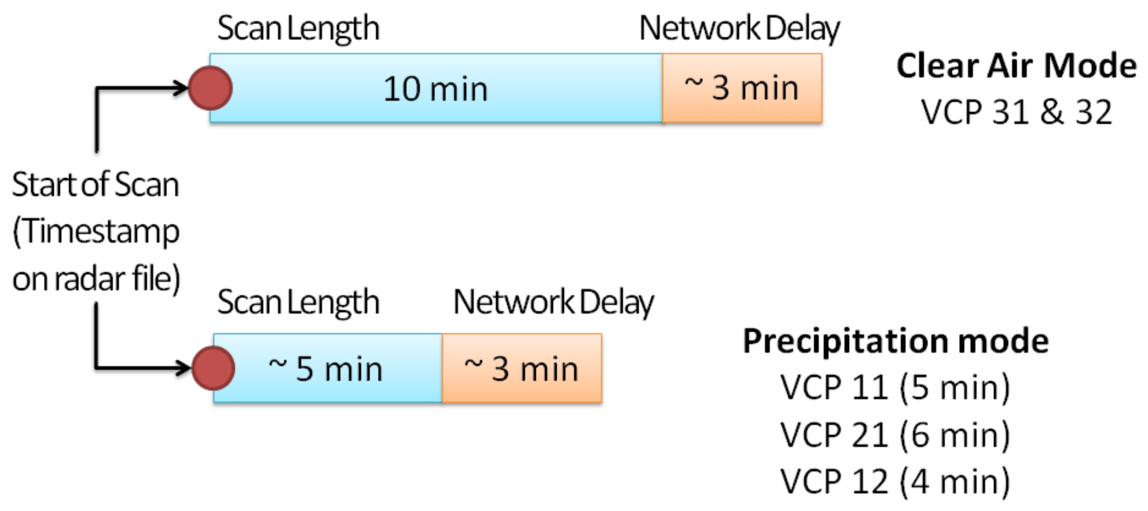


Figure 6.3: Timing uncertainties involved in reception of data using LDM for the two radar scanning-modes: In Clear-Air mode, scans are taken every 10 minutes on average. In Precipitation mode, scans are taken every 5 minutes on average. Metrics collected at our end also indicate a network-delay of about 3 minutes on average. It is important to note that there is nothing at our end that we can do to improve these values. The system design needs to be carried out with these as constraints.

- Even if files are received, they may be scrambled or partially-received and if they do not clear the Quality-Check, are equivalent to missing files.

Fortunately, the Module that is used for interpolation can interpolate for a time difference of anything lesser than 20 minutes. E.g. Radar data files from 10:03 and 10:17 can produce a files for 10:05, 10:10 & 10:15.

6.2.3 Trade-off: Quality of Product vs. Waiting Period

The upshot of the aforementioned uncertainties is that one has to wait longer to receive the next usable radar-file from which to generate interpolated products, which in turn can be merged with other such products to generate a Merged file. Figure 6.2.3 illustrates this situation.

As a product must be generated by a real-time deadline, the Module that merges

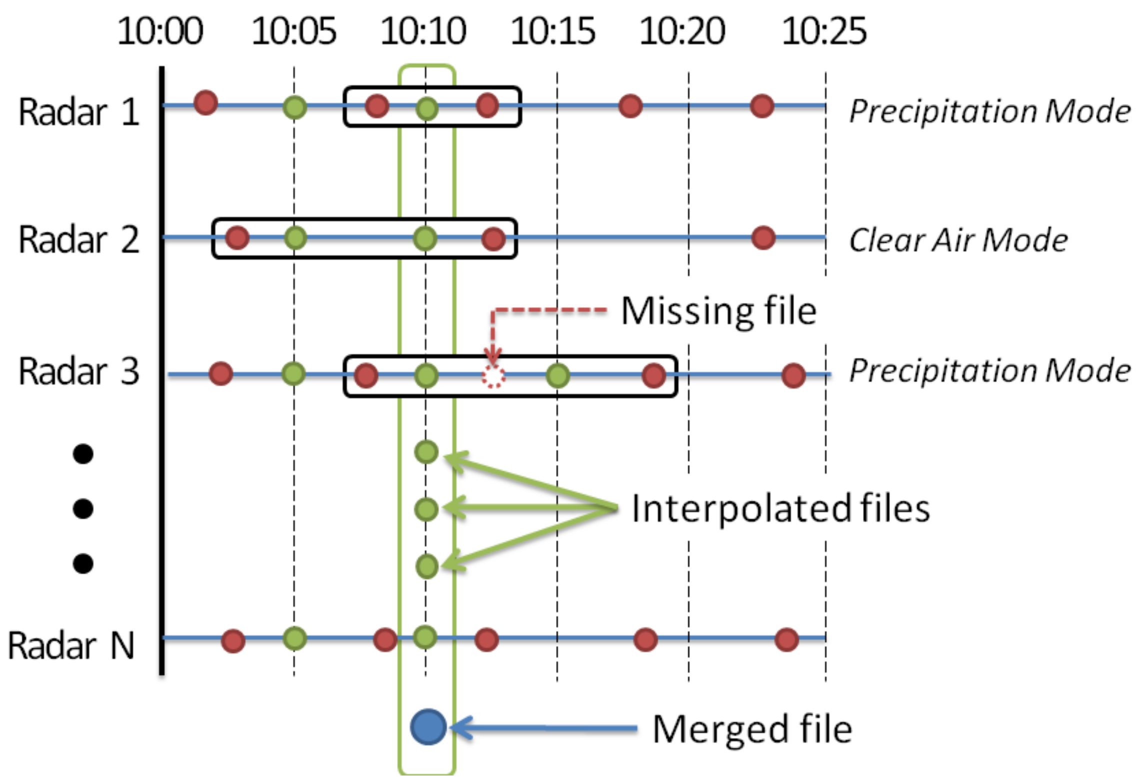


Figure 6.4: Multiple radar data synchronization in the case of missing files: To generate interpolated nominal-time files, one needs to radar-files that are adjacent in time. If, due to a reception-error, a file is not received at the expected time, one has to wait for a period equal to the current scanning period of that radar. With multiple radars involved in the generation of a merged-product, this could amount to a very long waiting period in the worst case.

the interpolated-files can be run with a (reduced) subset of interpolated-files. The resulting product could be incomplete i.e. have holes.

A simple trade-off has thus emerged: the longer the waiting-length, the greater the likelihood that the products being generated are more complete (i.e. generated from as many radars as possible from the total required). A greater delay, however, implies lower performance in a near real-time system.

In the extreme case that no radars from the required set are available, no product (or a blank product) is generated and this is reported in the latest.txt file in the output directory/URL.

6.2.4 Selection of Waiting Period

For an early prototype for the Iowa Flood Center, a feed was set up using seven radars that cover the State of Iowa. A metrics collection script was written to collect data on the availability of data from these radars for varying waiting intervals. Data gathered over a period of a few weeks showed the nature of the expected trade-off. An executive decision was made by the customer, Dr. Krajewski - Director, Iowa Flood Center, to select 25 minutes as the waiting window for that particular feed. For other feeds, this waiting period can be set manually.

The graph in figure 6.2.4 shows the variation in the number of radars for which data was available, for changes in the delay window (latency).

6.3 GUI - Feed-Manager interface

The Frontend/GUI needs to interact with the Feed-Manager on a processing machine for the following reasons:

- To set up a new feed
- To stop an existing feed
- To check a processing-machine's load

XML-RPC was chosen as the protocol for this interface for the following reasons:

- XML is human-readable, and therefore easy to compose manually and debug
- XML-RPC is simple to implement (relative to other web-services standards), and provides sufficient functionality
- XML-RPC libraries were easily available for both the technologies used in the GUI (PHP) and the Backend (Python)

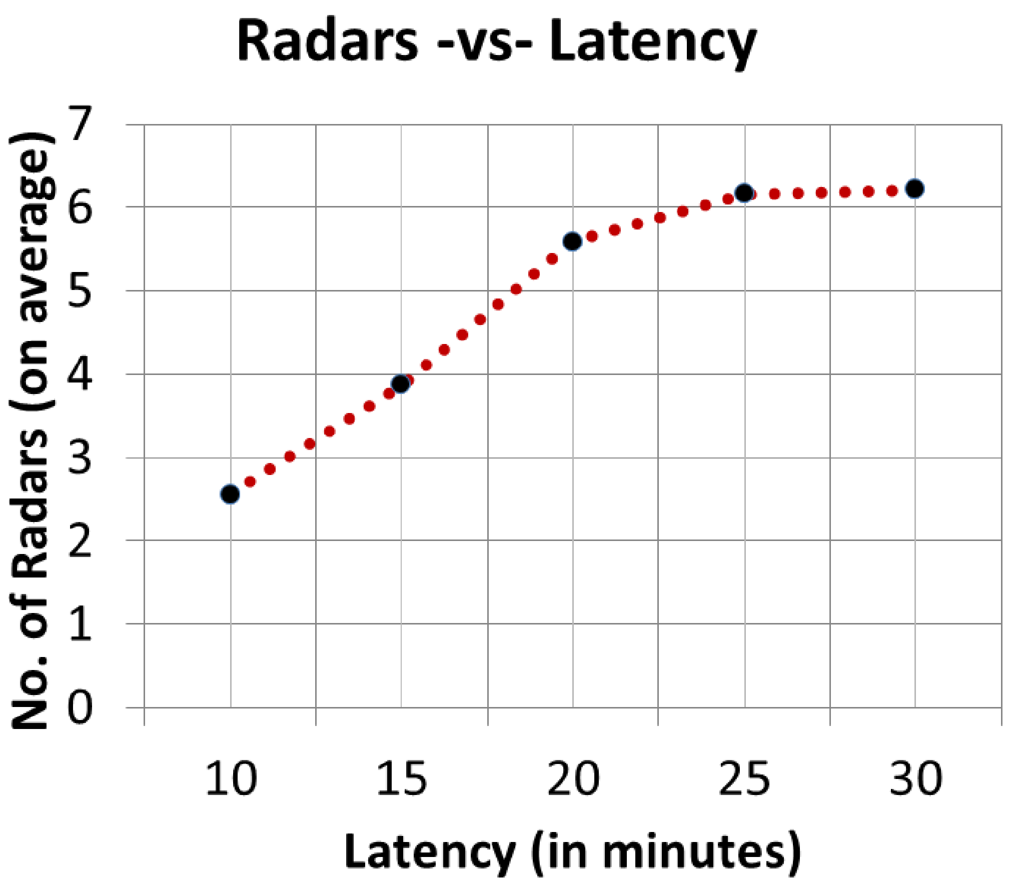


Figure 6.5: Quality vs. Latency Window Trade-off: The graph shows the variation in the number of radars for which data was available, for changes in the delay window (latency). The data point at 15-minutes/4-radars should be interpreted as: On average, data files from 4 radars (out of 7) is available for a delay window of 15 minutes, i.e. the merged rain-rate map that will be displayed will be from 15-minutes before current-time.

```
newSingleRadarFeed(user_email, radar, product, algorithm, grid, format,
debug_mode, pre_advection_correction, pre_product_type, pre_threshold,
pre_smoothing_width, ap_correction, pre_cappi, pre_cappi_height, pre_vpr,
rate_zr_scale, rate_hail_cap, rate_bin_size, rate_zr_power, accu,
accu_duration, accu_latency)
  Creates a new Single-Radar feed.
  Returns (Integer Status Code, Text String):
    If feed set up correctly, Status Code is 0. Text String is URL for Feed
    For all other Status Codes, Text String explains exception/error.

newMergingFeed(user_email, merge_type, radars, basin, product, algorithm,
grid, format, debug_mode, pre_advection_correction, pre_product_type,
pre_threshold, pre_smoothing_width, ap_correction, pre_cappi,
    pre_cappi_height, pre_vpr, rate_zr_scale, rate_hail_cap,
    rate_bin_size, rate_zr_power, accu, accu_duration, accu_latency)
  Creates a new Merging Feed
  Returns (Integer Status Code, Text String):
    If feed set up correctly, Status Code is 0. Text String is URL for Feed
    For all other Status Codes, Text String explains exception/error.

stopFeed(user_email, FeedID)
  Stops the feed identified by FeedID for the given user

getAvgLoadFactor()
  Returns the processors-averaged load averages 1-min, 5-min, 15-min
```

Table 6.2: API exposed by the Feed-Manager to the GUI

6.3.1 XML-RPC Application Programming Interface(API)

Table 6.3.1 shows the Application Programming Interface(API) that the Feed-Manager exposes to the GUI.

6.3.2 Notes on the API

The API was designed with the following in mind:

- The API was only meant to be used by the HNX2 Frontend/GUI, and not meant to be made public. Therefore, no API-level security/authentication features were required. For security, the firewall on the XML-RPC server machine could be configured to receive connections from only the GUI machine or the GUI and XML-RPC Server could share a secret-key.

- The primary purpose of the API was to provide a good boundary of separation between the Frontend/GUI and any Feed-Manager machine. This is important to allow independent development of the Frontend/GUI and Backend. As mentioned in section 3.2, the Frontend may be used for multiple projects and has a developer dedicated to it.
- Minimizing the number of functions exposed by the XML-RPC server is also recommended as a general design practice [36]. The API should provide the simplest yet complete interface for all required features. The only reason why there are two functions in the API for starting new feeds, one for single-radar feeds and one for multiple-radar feeds, is because they have significantly different parameter-lists. This reduces programming errors when using the API.
- The exhaustiveness of the parameter list for new-feed procedure calls is explained as follows:
  - Standard Software Engineering practices normally dictate a strict separation between the "GUI/Presentation" layer and "Logic" layer [37], to extend the longevity of the system. By sufficiently decoupling the different layers, one can eliminate the the need to make changes in any one layer of the system when the other is changed.
  - This constraint has been somewhat compromised in the design of HNX2 in that the HNX2 Frontend/GUI does contain some "logic" in it. However, as the following explanation shows, this is done to extend the longevity of the system.

- As mentioned in section 3.2, a dedicated full-time employee of IIHR manages the development of the HNX2 GUI (and other GUIs). Given the frequency of algorithmic changes being made on the system's underlying processing Modules (as expected in a research environment) it seemed best to not have to modify the Backend Python scripts each time a change is required.
- For this reason, the HNX2 GUI translates algorithm keywords (like 'quick-look', 'pseudopps' and 'hifi'; see section 2.6) into their 'custom' algorithm equivalents. The GUI thus stores a look-up table of algorithm parameters (e.g. pre_threshold, rate_zr_scale, ...) for each non-custom parameter. The algorithm parameter passed through the XML-RPC call is redundant, and not used by the Backend.
- This arrangement permits small tweaks to be made to the processing options for the various algorithms with minor changes to the calling XML-RPC call's arguments but without requiring any changes to the API or the Backend's Python scripts.

6.4 Feed Configuration file

The feed-configuration file (*feed.conf*) encapsulates all the data needed by the Feed-Processor in order to generate the new product. This design allows the Feed-Processor to function independently without needing to access the database on each instantiation.

The feed-configuration file is generated automatically by the Feed-Manager when an XML-RPC request to set up a new feed is recevied from the GUI. It consists of key-value entries required by either the Feed-Processor script (e.g. location of Harvested radar-data files, location of Modules) or by individual Modules (e.g. values of arguments for various algorithm parameters, basin-related or radar-related metadata). The Feed-Manager generates the feed-configuration file using the parameters passed to it by the GUI through the XML-RPC call and by querying the HNX2 database to get basin-related (e.g. basin area, basin bounding-box) or radar-related metadata (e.g. radar height, latitude, longitude).

The format of the file is plain-text and in a human-readable Windows INI-file like format. This allows for manual tweaks to the file if required or for setting up a feed completely independent of the standard GUI-based process.

Examples of feed-configuration files follow:

6.4.1 Feed Configuration file for Single Radar feeds

Refer to table 6.4.1. Note that entries like like "pre_radar_tower" and "radar_latlon" have been extracted from the HNX2 Database, whilst entries like "algorithm" and "ap_correction" are taken from the XML-RPC call from the GUI.

6.4.2 Feed Configuration file for Multiple Radar feeds

Refer to table 6.4.2. Note that the sections on radars "[KDVN]" and "[KDMX]" and on the basin "[BASIN]" have been extracted from the HNX2 Database, whilst most other entries are taken from the XML-RPC call from the GUI. Note that the "radars" parameter of the XML-RPC call is a string of variable length.

6.5 Scalability

The amount of data being processed by the HNX2 is significant: Each one of the NEXRAD radars on average produces one file every 4-5 minutes during times of activity and every 10 minutes when there is no precipitation activity. The size of each file varies with the local atmospheric conditions captured by each scan, but on average is about 2.5 MB in a compressed format. Once received, the file is re-sequenced, uncompressed, quality-checked, format-converted, compressed and transferred over the network to predefined locations on all active feed-processing machines. On the feed-processing machines, data is sent through the data-processing pipeline which runs

```
[hnx2]
module_folder = /home/nexrad/HNX2/modules/
harvest = /home/nexrad/harvest/

[feed]
outputpath = /var/www/html/feeds/SR/test
radar = KDVN
type = single-radar
algorithm = pseudo_pps
product = accumulation

[sr_pre]
pre_advection_correction = off
pre_threshold = 2
pre_smoothing_width = 1.0
pre_product_type = 10
ap_correction = on
pre_radar_tower = 20
pre_cappi_height = 2.0
pre_vpr = on

[sr_rate]
rate_zr_scale = 300
rate_hail_cap = 55
rate_bin_size = 1
rate_zr_power = 1.4

[sr_accu]
accu = on
accu_running = off
accu_duration = 15
accu_latency = 0

[sr_grid]
grid = latlon
radar_latlon = 41.611 -91.58
radar_polar_bb = 0 360 0 230
radar_laton_bb = 41.4 -91.3 39.2 -87.5

[sr_format]
format = arcascii
```

Table 6.3: Feed Configuration file for Single Radar feeds

```
[feed]
type = data-merging
outputpath = /var/www/html/feeds/DM/test
radars = KDVN;KDMX;
basin = 0106
algorithm = custom

[dm_stage1]
mpre_threshold = 2
mpre_smoothing_width = 1.0
mpre_product_type = 10
ap_correction = on
mpre_radar_tower = 20
mpre_cappi_height = 2.0

[dm_stage2]
rate_zr_scale = 300
rate_hail_cap = 55
rate_bin_size = 1
rate_zr_power = 1.4

[dm_stage3]
accu = on
accu_duration = 15

[dm_grid]
grid = latlon

[dm_format]
format = arcascii

[KDVN]
latlon = 41.611 -91.58
height = 229.82 25

[KDMX]
latlon = 41.73 -93.723
height = 299 30

[BASIN]
polar_bb = 0 360 0 230
basin_boundingbox = (45.531224,-70.840001,44.913616,-69.940929)
```

Table 6.4: Feed Configuration file (abridged) for multiple-radar Data-Merging feeds

them through hydrological algorithms, time-synchronizes and merges files if working on data from multiple radars, and finally converts the processed data to the format of the user's choice.

The ambient CPU load of such activities is significant in normal conditions, and is known to rise to 2-3 times the ambient conditions load during heavy precipitation activity. Disk-space footprints are as follows: A product being produced from a set of 7 radars that cover the state of Iowa has, on average, a size of 12 MB in ASCII-text format. The processing footprint of a job is significantly larger, averaging about 150 MB of disk space per radar involved in the job.

There are two distinct types of scalability that apply to HNX2:

- **LDM scalability**: This refers to the ability of the system to scale to handle the reception of data from a large number of NEXRAD radars.
- **Feeds scalability**: This refers to the ability of the system to scale to handle an arbitrarily large number of user-customized feeds.

In both types, our solutions require the addition of new machines to distribute the additional workload that comes from scaling up. A fully automated scaling process was not required as the process of adding new machines requires planning and executive approval at the IIHR. This complies with the requirement stated in section 3.1.

6.5.1 Measuring system load

In Linux/Unix, the 'load' on a system is tracked by the kernel in a parameter referred to as "Load Number". Specifically, the Load Number is the number of processes waiting in ready-state (or run queue) to be scheduled for execution on the CPU [38].

Linux reports the load number in the form of 3 numbers called Load Average triplets, which are the weighted moving average of the load number for the past 1, 5 and 15 minutes of operation. [38]

In machines with multiple-cores, the Load Factor needs to be divided by the number of cores to get the Average Load Factor which is a more accurate measure when comparing different machines. There is no well-established load number which is considered ideal for safe performance. In general, however, a processor-averaged load average greater than 1 indicates clearly that a machine is overloaded.

6.5.2 LDM Scalability

The LDM daemon, format-conversion & quality-conversion scripts and compression add up to significant processing load on a system. Additionally, load goes up 2-3 times the ambient conditions load during precipitation activity.

To scale this setup up, we can distribute the reception of radar-data to multiple machines. The Harvester gets a list of processing-machines to which it needs to deposit usable radar-files to from the database. This simple arrangement ensures that all processing-machines receive data from the entire set of radars.

6.5.3 Feeds Scalability

HNX2 has been designed to serve and arbitrarily large number of user feeds.

When a user sets up a new feed, the GUI returns him a URL where he can expect new files to be available. When setting a new feed up the GUI gets a list of all active processing-machines from the database. It then makes an XML-RPC call to each machine to get it's Average Load Factor. It then sets up the new feed on the machine with the least Average Load Factor. This simplistic protocol allows us to balance feed-processing load between multiple processing-machines and add new machines as the number of feeds that need to be run increases.

For both LDM Scalability and Feeds Scalability, manual intervention is required to set up new machines. HNX2 System administrators can monitor the load on each machine in the system and make preemptive scaling decisions before systems get overloaded.

CHAPTER 7
DISCUSSION

Overall, the development of HNX2 was a multifaceted software-engineering challenge, requiring problem-solving at many abstraction levels (Figure 7).

An agile process was followed to allow the system to evolve over the period of development. A high level scripting language, Python, was used to rapidly prototype the components of the system and to conduct engineering experiments.

At lower levels of abstraction, we faced issues like high error-rate due to partially received files from LDM. This led to an investigation into the format of the files received through LDM, and resulted in a changes being made to the format-converter module and the development of quality-check and partial-file format-converter modules.

At a higher level, experiments were carried out to determine an acceptable waiting-time for availing real-time data-products made by merging data from multiple radars.

Contrary to standard software-engineering practices, some logic/intelligence was left on the system Frontend (which really is more than a GUI) to extend the longevity of the system and to suit the organizational structure at the IIHR.

The system has been designed to be scale to receive data from as many NEXRAD radars (from within the continental US) as required and produce as many user-customized feeds as required.

7.1 Prototyping

An end-to-end prototype was made relatively early in the project: A feed producing rain-rate maps (every 5 minutes) and accumulation maps (every 15-minutes), generated by merging data from seven-radars (Call signs: KDVN, KDMX, KMPX, KARX, KOAX, KFSD, KEAX) covering the state of Iowa was set up. This data drives flood-forecasting CUENCAS models [30] being used in the research of the Iowa Flood Center, which is housed at the IIHR. Based on their requirements and

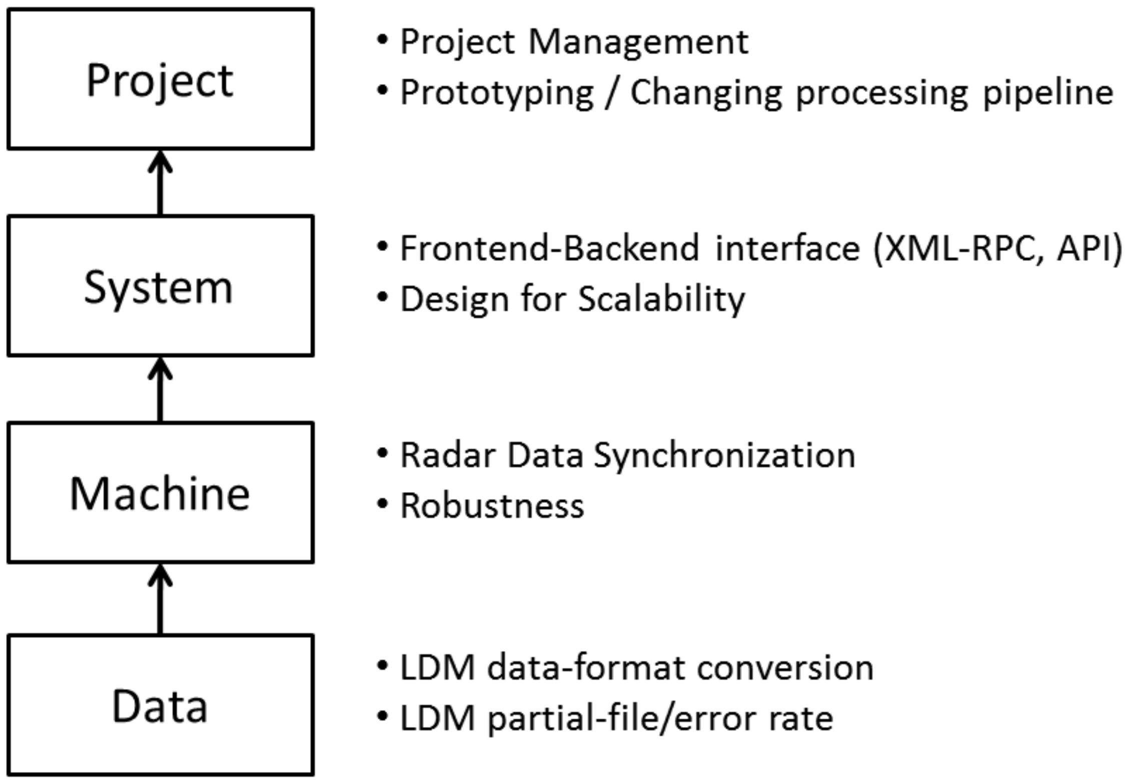

Project
System
Machine
Data
• Project Management
• Prototyping / Changing processing pipeline
• Frontend-Backend interface (XML-RPC, API)
• Design for Scalability
• Radar Data Synchronization
• Robustness
• LDM data-format conversion
• LDM partial-file/error rate

Figure 7.1: Challenges at multiple levels of abstraction

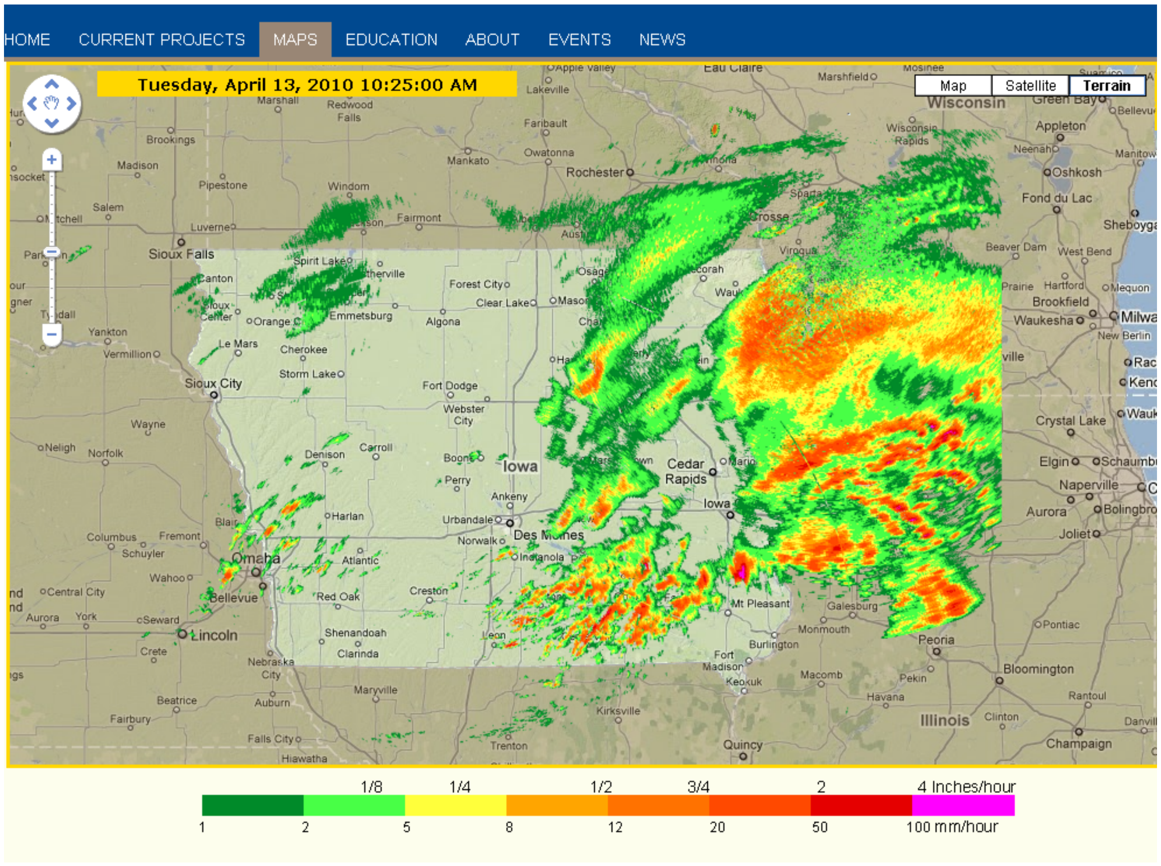


Figure 7.2: Google-Maps based visualization of the sever-radar Iowa Flood Center rain-rate map feed

feedback on the quality of the products, some changes to the underlying processing pipeline were made and are still being made. This often involved setting up test-feeds and other testing-related arrangements, and also meant providing project-level support. A Google-Maps based visualization of 5-minute rain-rate maps, as shown in figure 7.1, was made available to the public through the Iowa Flood Center website: `http://www.iowafloodcenter.org/maps/`

7.2 State of development efforts

At the time of submission of this manuscript:

- All of the larger components of the system (viz. GUI, Harvesters, Feed-Manager, Feed-Processor) have been coded.

- End-to-end automation (i.e. feed configuration through the GUI) is available for single-radar feeds, but not for merging feeds because of significant recent changes made to the pipeline.

- Merging feeds can currently be manually set up. We have been running the following feeds:

  - The seven-radar (Call signs: KDVN, KDMX, KMPX, KARX, KOAX, KFSD, KEAX) Iowa Flood Center feed (see sections 7.1 and 3.2) has been running since February 2010. Numerous changes and tweaks have been made since it was first set up. It delivers rain-rate maps every 5-minutes and accumulation-maps every 15-minutes, both covering the entire state of Iowa.

  - The three-radar (Call Signs: KLOT, KIND, KILX) feed for Prof. Barbara Minsker's group at UIUC [39] has been running since June 2010. It delivers rain-rate maps every 5-minutes, covering the Upper Embarras Watershed in Illinois.

  - Numerous short-lived test feeds were set up from time to time during the development of HNX2 to carry out end-to-end (from GUI to product-delivery) testing of the system.

7.3 Conclusion

Challenges were encountered at multiple levels of abstraction, and were overcome. A design that met all desiderata, and a prototype that successfully implemented the design from end-to-end, were completed.The prototype is an instance of a generalized scalable framework that runs on standard hardware and can provide user-customized products for basins anywhere in the continental United States.

HNX2 Homepage: http://www.hydro-nexrad.net/

REFERENCES

[1] National Weather Service, “National Doppler Radar Sites,” http://radar.weather.gov/ (accessed Sept. 25, 2010).

[2] The Weather Channel, “About Weather.com,” http://www.weather.com/aboutus/ (accessed Sept. 25, 2010).

[3] Radar Operations Center, Norman, OK, “NEXRAD WSR-88D,” http://www.roc.noaa.gov/WSR88D/Images/WSR-88DCONUSCoverage1000.jpg (accessed Sept. 25, 2010).

[4] Radar Operations Center, Norman, OK, “About the ROC,” http://www.roc.noaa.gov/WSR88D/About.aspx (accessed Sept. 25, 2010).

[5] NOAA NWS, “Doppler Radar Frequently Asked Questions,” http://www.srh.noaa.gov/srh/sod/radar/radinfo/radinfo.html (accessed Sept. 25, 2010).

[6] NWS JetStream, “Introduction to Doppler Radar,” http://www.srh.noaa.gov/jetstream/doppler/doppler_intro.htm (accessed Sept. 25, 2010).

[7] WeatherTAP, “NEXRAD Radar Tutorial,” http://www.weathertap.com/guides/radar/weather-radar-tutorial.html (accessed Sept. 25, 2010).

[8] NOAA Satellite and Information Service, “NCDC Radar Products,” http://www.ncdc.noaa.gov/oa/radar/radarproducts.html (accessed Sept. 25, 2010).

[9] NEXRAD Radar Operations Center, “WSR-88D Build 10/Super Resolution Level II FAQs,” http://www.roc.noaa.gov/wsr88d/buildinfo/build10faq.aspx (accessed Sept. 25, 2010).

[10] NOAA Satellite and Information Services, “NCDC: Surface Documentation, Dataset 6500: NEXRAD Level II High Resolution Base Data,” http://www.ncdc.noaa.gov/oa/documentlibrary/surface-doc.html#6500 (accessed Sept. 25, 2010).

[11] NOAA Satellite and Information Service, “NCDC: Surface Documentation, Dataset 7000: NEXRAD Level III Products,” http://www.ncdc.noaa.gov/oa/documentlibrary/surface-doc.html#7000 (accessed Sept. 25, 2010).

[12] NOAA Satellite and Information Service, “NEXRAD/TDWR Level-III Products,” http://www.ncdc.noaa.gov/oa/radar/productsdetail.html (accessed Sept. 25, 2010).

[13] Unidata Program Center, “About Unidata,” http://www.unidata.ucar.edu/about/ (accessed Sept. 25, 2010).

[14] Unidata Program Center, “Local Data Manager (LDM),” http://www.unidata.ucar.edu/software/ldm (accessed Sept. 25, 2010).

[15] Unidata Program Center, “Unidata Internet Data Distribution: An Overview of the IDD,” http://www.unidata.ucar.edu/software/idd/overview/idd.html (accessed Sept. 25, 2010).

[16] Ben Domenico, Sally Bates and Dave Fulker, Unidata Program Center, “Internet Data Distribution (IDD),” http://www.unidata.ucar.edu/software/idd/iddams.html (accessed Sept. 25, 2010).

[17] Unidata Program Center, “LDM Factsheet,” http://www.unidata.ucar.edu/software/ldm/factsheet.html (accessed Sept. 25, 2010).

[18] NOAA Satellite and Information Service, “About NCDC,” http://www.ncdc.noaa.gov/oa/about/about.html (accessed Sept. 25, 2010).

[19] NOAA Satellite and Information Service, “NCDC NEXRAD Data Inventory Search,” http://www.ncdc.noaa.gov/nexradinv (accessed Sept. 25, 2010).

[20] NOAA Satellite and Information Service, “NCDC: Decode Radar Data,” http://lwf.ncdc.noaa.gov/oa/radar/radardata.html (accessed Sept. 25, 2010).

[21] NASA TRMM Satellite Validation Office, “TRMM Radar Software Library,” http://trmm-fc.gsfc.nasa.gov/trmm_gv/software/rsl/ (accessed Sept. 25, 2010).

[22] NASA Tropical Rainfall Measuring Mission, “About TRMM,” http://trmm.gsfc.nasa.gov/overview_dir/background.html (accessed Sept. 25, 2010).

[23] W. F. Krajewski, B.-C. Seo, A. Kruger, P. Domaszczynski, G. Villarini, and C. Gunyon, “Hydro-nexrad radar-rainfall estimation algorithm development, testing and evaluation,” K. C. Kabbes, Ed., vol. 243, no. 40927. ASCE, 2007, pp. 279–279. [Online]. Available: http://link.aip.org/link/?ASC/243/279/1

[24] W. F. Krajewski, A. Kruger, R. Lawrence, J. A. Smith, A. A. Bradley, M. Steiner, M. L. Baeck, M. K. Ramamurthy, J. Weber, S. A. DelGreco, B.-C. Seo, P. Domaszczynski, C. Gunyon, and R. Goska, “Towards better utilization of nexrad data in hydrology: An overview of hydro-nexrad,” K. C. Kabbes, Ed., vol. 243, no. 40927. ASCE, 2007, pp. 288–288. [Online]. Available: http://link.aip.org/link/?ASC/243/288/1

[25] A. Kruger and W. F. Krajewski, "Efficient storage of weather radar data," *Softw. Pract. Exper.*, vol. 27, no. 6, pp. 623–635, 1997.

[26] A. Kruger, R. Lawrence, and E. Dragut, "Building a terabyte nexrad radar database for hydrometeorology research," *Computers & Geosciences*, vol. 32, no. 2, pp. 247 – 258, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/B6V7D-4GSJR67-2/2/d94c0bbcc8dfbbbc88641dac819fc06e

[27] W. F. K. Anton Kruger and P. Domaszczynski, "Hydro-nexrad: metadata computation and use," *Journal of Hydroinformatics*, vol. In Press. [Online]. Available: http://www.iwaponline.com/jh/up/hydro2010057.htm

[28] E. Douglas Jensen, "Hard and Soft Real-Time — Real-Time for the Real World," http://www.real-time.org/real-time/hard-and-soft-real-time.html (accessed Sept. 25, 2010).

[29] US Department of Defense, "Dictionary of Military and Associated Terms: near real-time," http://www.dtic.mil/doctrine/dod_dictionary/data/r/6530.html/ (accessed Sept. 25, 2010).

[30] R. Mantilla and V. K. Gupta, "A GIS framework to investigate the process basis for scaling statistics on river networks," *Geoscience and Remote Sensing Letters, IEEE*, vol. 2, no. 4, pp. 404–408, 2005.

[31] Y. Liu, D. J. Hill, A. Rodriguez, L. Marini, R. Kooper, J. Futrelle, B. Minsker, and J. D. Myers, "Near-real-time precipitation virtual sensor using nexrad data," in *GIS '08: Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems.* New York, NY, USA: ACM, 2008, pp. 1–2.

[32] Red Hat Inc., "Red Hat Enterprise Linux," http://www.redhat.com/rhel/ (accessed Sept. 25, 2010).

[33] American Geophysical Union, "(Session on) Integrated Modeling and Python Applications in Hydrology II," http://www.agu.org/meetings/fm08/fm08-sessions/fm08_H51L.html (accessed Sept. 25, 2010).

[34] NOAA Satellite and Information Services, "Decode NEXRAD data to NetCDF," http://www.ncdc.noaa.gov/oa/radar/radar-decoding-easy.html (accessed Sept. 25, 2010).

[35] National Weather Service, "NWS Enhanced Radar Image, Quad Cities, IA Radar," http://radar.weather.gov/radar.php?rid=dvn (accessed Sept. 25, 2010).

[36] J. Bloch, "How to design a good api and why it matters," in *OOPSLA '06: Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications.* New York, NY, USA: ACM, 2006, pp. 506–507.

[37] Paragon Corporation, "Separation of Business Logic from Presentation Logic in Web Applications," http://www.paragoncorporation.com/ArticleDetail.aspx?ArticleID=21 (accessed Sept. 25, 2010).

[38] Dr. Neil Gunther, Teamquest Corporation, "UNIX Load Average Part 1: How It Works," http://www.teamquest.com/resources/gunther/display/5/index.htm (accessed Sept. 25, 2010).

[39] University of Illinois, Urbana-Champaign, "Barbara S. Minsker — Civil and Environmental Engineering at Illinois," http://cee.illinois.edu/node/120 (accessed Sept. 25, 2010).